République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE MUSTAPHA STAMBOULI DE MASCARA

FACULTÉ DES SCIENCES ET TECHNOLOGIE

Polycopié de Cours

Logiciels de simulation

Présenté par **:**

Dr: AFIF BENAMEUR

Ce polycopié de cours destiné aux étudiants de Parcours LMD de licence
ELECTROTECHNIQUE
3ème Année / Semestre : 05

Algérie
2021/2022

Avant-propos

Ce polycopié de cours destiné aux étudiants LMD de licence 3ème année spécialité Electrotechnique, correspond au programme officiel du Ministère de l'Enseignement Supérieur et de la Recherche Scientifique pour le module Logiciel de simulations. Ce polycopié est organisé en 9 chapitres.

Table de matières

INTRODUCTION

Introduction

MATLAB est une abréviation de Matrix LABoratory. Écrit à l'origine, en Fortran, par C. Moler, MATLAB était destiné à faciliter l'accès au logiciel matriciel développé dans les projets LINPACK et EISPACK. La version actuelle, écrite en C par the MathWorks Inc., existe en version "professionnelle" et en version "étudiant". MATLAB est conforté par une multitude de boîtes à outils (Toolboxes) spécifiques à des domaines variés. Un autre atout de MATLAB, est sa portabilité ; la même portion de code peut être utilisée sur différentes plates-formes sans la moindre modification.

MATLAB est un environnement puissant, complet et facile à utiliser destiné au calcul scientifique. Il apporte aux ingénieurs, chercheurs et à tout scientifique un système interactif intégrant calcul numérique et visualisation. C'est un environnement performant, ouvert et programmable qui permet de remarquables gains de productivité et de créativité.

MATLAB est un environnement complet, ouvert et extensible pour le calcul et la visualisation. Il dispose de plusieurs centaines (voire milliers, selon les versions et les modules optionnels autour du noyeau Matlab) de fonctions mathématiques, scientifiques et techniques.

L'approche matricielle de MATLAB permet de traiter les données sans aucune limitation de taille et de réaliser des calculs numériques et symboliques de façon fiable et rapide. Grâce aux fonctions graphiques de MATLAB, il devient très facile de modifier interactivement les différents paramètres des graphiques pour les adapter selon nos souhaits.

L'approche ouverte de MATLAB permet de construire un outil sur mesure. On peut inspecter le code source et les algorithmes des bibliothèques de fonctions (Toolboxes), modifier des fonctions existantes et ajouter d'autres.

MATLAB possède son propre langage, intuitif et naturel qui permet des gains de temps de CPU spectaculaires par rapport à des langages comme le C, le TurboPascal et le Fortran. Avec MATLAB, on peut faire des liaisons de façon dynamique, à des programmes C ou Fortran, échanger des données avec d'autres applications (via la

DDE : MATLAB serveur ou client) ou utiliser MATLAB comme moteur d'analyse et de visualisation.

MATLAB comprend aussi un ensemble d'outils spécifiques à des domaines, appelés Toolboxes (ou Boîtes à Outils). Indispensables à la plupart des utilisateurs, les Boîtes à Outils sont des collections de fonctions qui étendent l'environnement MATLAB pour résoudre des catégories spécifiques de problèmes. Les domaines couverts sont très variés et comprennent notamment le traitement du signal, l'automatique, l'identification de systèmes, les réseaux de neurones, la logique floue, le calcul de structure, les statistiques, etc.

MATLAB fait également partie d'un ensemble d'outils intégrés dédiés au Traitement du Signal. En complément du noyau de calcul MATLAB, l'environnement comprend des modules optionnels qui sont parfaitement intégrés à l'ensemble :

1) une vaste gamme de bibliothèques de fonctions spécialisées (Toolboxes)

2) Simulink, un environnement puissant de modélisation basée sur les schémas-blocs et de simulation de systèmes dynamiques linéaires et non linéaires

3) Des bibliothèques de blocs Simulink spécialisés (Blocksets)

4) D'autres modules dont un Compilateur, un générateur de code C, un accélérateur,...

5) Un ensemble d'outils intégrés dédiés au Traitement du Signal : le DSP Workshop.

CHAPITRE 1

INTRODUCTION A L'ENVIRONNEMENT MATLAB

1 Chapitre 01: INTRODUCTION A L'ENVIRONNEMENT MATLAB

1.1 INTRODUCTION :

MATLAB (MATrix LABoratory) est une atmosphère de programmation instructive formative pour le calcul scientifique, la programmation et la visualisation des données.

Son utilité est importante dans les domaines d'ingénierie et de recherche scientifique, ainsi qu'aux établissements d'enseignement supérieur. Sa popularité est due principalement à sa forte et simple interaction avec l'utilisateur mais aussi aux points suivants :

- Sa fortune aisance fonctionnelle : avec MATLAB licite de raviser des manipulations mathématiques complexes en écrivant peu d'instructions. Evolue des expressions, dessiner des graphiques et exécuter des programmes classiques. Et surtout, avec une utilisation directe meilleurs fonctions prédestiné
- Un des cas d'un choix l'utilisation les boites à outils (toolboxes) : encourage plusieurs disciplines (simulation, traitement de signal, imagerie, intelligence artificielle,...etc.).
- L'ingénuité de son langage de programmation : un programme écrit en MATLAB est usuel plaisant à écrire et à lire comparé au même programme écrit en C ou en PASCAL
- En particulier de tout gérer comme étant des matrices, ce qui libère l'utilisateur de s'occuper de typage de données d'éloigner les problèmes de transtypage

Provence MATLAB conçu pour faire principalement des calculs sur les vecteurs et les matrices d'où son nom '**Mat**rix **Lab**oratory' , a augmenter les performances et affermer négocier plus de domaines.

MATLAB n'est pas le seul environnement de calcul scientifique

existant puis ce qu'il existe d'autres adversaires dont les plus importants sont Maple et Mathématique. Il Ya même des logiciels libres qui sont des clones de Matlab comme Scilab et Octave.

1.2 L'ENVIRONNEMENT MATLAB

MATLAB affiche plusieurs fenêtres. Selon la version on peut trouver les fenêtres suivantes :

- ***Current Folder:*** indique le répertoire courant ainsi que les fichiers existants.
- ***Workspace:*** indique toutes les variables existantes avec leurs types et valeurs.
- ***Command History***: garde la trace de toutes les commandes entrées par l'utilisateur.
- ***Command Window***: L'une des plus importantes fenêtres de Matlab, la Command Window traite des instructions données. C'est après l'invite ("prompt") » qu'il faut entrer les instructions demandées

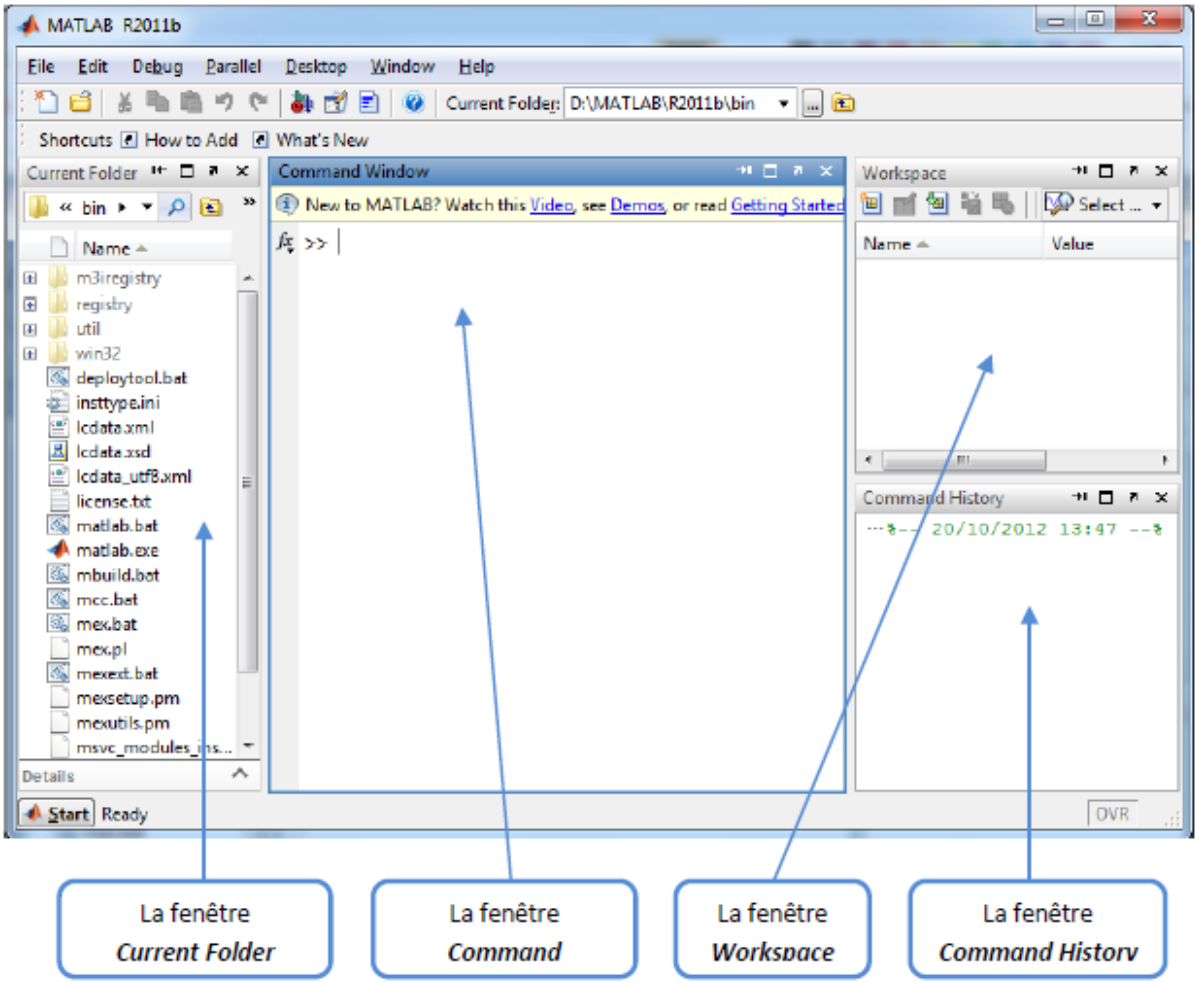


Figure 1.1 : L'environnement MATLAB (Version *2011b* ou *7.13*)

1.2.1 Première interaction avec MATLAB:

Le moyen le plus simple d'utiliser MATLAB est d'écrire directement dans la fenêtre de commande (*Command Window*) juste après le curseur (prompt) >>

Pour calculer une expression mathématique il suffit de l'écrire comme ceci :

```
>> 15+17                    Puis on clique sur la touche Entrer pour voir le résultat
      ans =
            32
```

Si nous voulons qu'une expression soit calculée mais sans afficher le résultat, on ajoute un point-virgule '**;**' à la fin de l'expression comme suit :

```
>> 15+17 ;
>>
```

Pour créer une variable on utilise la structure simple : 'variable = définition' sans se préoccuper du type de la variable.

Exemple :

```
>> a = 11 ;
>> u = cos(a) ;
>> v = sin(a) ;
>>u^2+v^2
      ans =
            1
>> ans+10
      ans =
            12
>>
```

Il est possible d'écrire plusieurs expressions dans la même ligne en les faisant séparées par des virgules(,) ou des points virgules (;). Par exemple :

```
>> 11+8, 3*8-4, 15-9

      ans =
            17
      ans =
            20
      ans =
            6
>> 11+8; 3*8-4, 15-9;
      ans =
            20
>>
```

Le nom d'une variable ne doit contenir que des caractères alphanumériques ou le symbole '_' (underscore), et doit commencer par un alphabet. Nous devons aussi faire attention aux majuscules car le MATLAB est sensible à la casse (***A*** et ***a*** sont deux identifiants différents).

Les opérations de base dans une expression sont résumées dans le tableau suivant :

L'opération	La signification
+	L'addition
-	La soustraction
*	La multiplication
/	La division
\	La division gauche (ou la division inverse)
^	La puissance
'	Le transposé

Pour voir la liste des variables utilisées, soit on regarde à la fenêtre

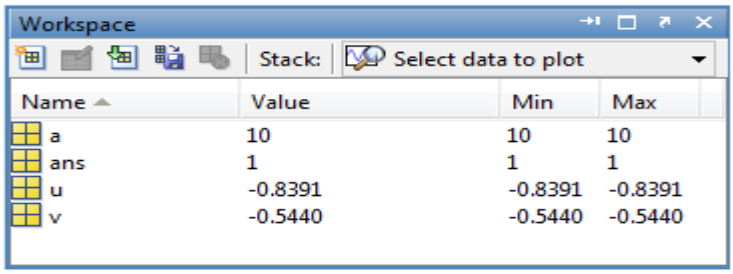
'***Workspace'*** soit on utilise les commandes '***whos'*** ou '***who'***.

whos donne une description détaillée (le nom de la variable, son type et sa taille), par contre ***who*** donne juste les noms des variables.

Par exemple, dans ce cours on a utilisé 3 variables ***a***, ***u*** et ***v***:

```
>>who
      Your variables are:
      a    ans  u    v

>>whos
      Name        Size              Bytes  Class     Attributes
      a           1x1                   8  double
      ans         1x1                   8  double
      u           1x1                   8  double
      v           1x1                   8  double
```

L'utilisation de ces deux commandes peut être omise car des informations sur les variables sont visibles directement dans la fenêtre *workspace*.



1.2.2 Les nombres en MATLAB :

MATLAB utilise une notation décimale conventionnelle, avec un point décimal facultatif '**.**' et le signe '**+**' ou '**–**' pour les nombres signés. La notation scientifique utilise la lettre '***e***' pour spécifier le facteur d'échelle en puissance de **10**. Les nombres complexes utilise les caractères '**i**' et '**j**' (indifféremment) pour designer la partie imaginaire. Le tableau suivant donne un résumé :

Le type	Exemples
Entier	5 -83
Réel en notation décimale	0.0205 3.1415926
Réel en notation scientifique	1.60210e-20 6.02252e23 (1.60210×10^{-20} et 6.02252×10^{23})
Complexe	5+3i -3.14159j

MATLAB utilise toujours les nombres réels (double précision) pour faire les calculs, ce qui permet d'obtenir une précision de calcule allant jusqu'aux 16 chiffres significatifs, mais il faut noter les points suivants :

- Le résultat d'une opération de calcule est par défaut affichée avec quatre chiffres après la virgule.
- Pour afficher davantage de chiffres utiliser la commande *format long* (14 chiffres après la virgule).
- Pour retourner à l'affichage par défaut, utiliser la commande *format short.*
- Pour afficher uniquement 02 chiffres après la virgule, utiliser la commande *format bank*.
- Pour afficher les nombres sous forme d'une ration, utiliser la commande *format rat.*

La commande	Signification
format short	affiche les nombres avec 04 chiffres après la virgule
format long	affiche les nombres avec 14 chiffres après la virgule
format bank	affiche les nombres avec 02 chiffres après la virgule
format rat	affiche les nombres sous forme d'une ration (a/b)

Exemple :

```
>> 7/3
       ans =
              2.3333

>>format long

>> 7/3
       ans =
              2.333333333333334
>>format bank

>> 7/3
       ans =
              2.33
>>format short

>> 7/3
       ans =
              2.3333

>> 5.4*6.7
       ans =
              36.1800

>>format rat

>> 5.4*6.7
       ans =
              1809/50
```

```
>> 36.1800
ans =
   1809/50
```

La fonction **vpa** peut être utilisé afin de forcer le calcule de présenter plus de décimaux significatifs en spécifiant le nombre de décimaux désirés.

Exemple :

```
>>sqrt(2)
      ans =
            1.4142

>>vpa(sqrt(2),27)
      ans =
            1.41421356237309504880168872
```

1.2.3 Les principales constantes, fonctions et commandes:

MATLAB définit les constantes suivantes :

La constante	Sa valeur
pi	π=3.1415…
exp(1)	e=2.7183…
i	$=\sqrt{-1}$
j	$=\sqrt{-1}$
Inf	∞
NaN	Not a Number (Pas un numéro)
eps	$\varepsilon \approx 2 \times 10^{-16}$

Parmi les fonctions fréquemment utilisées, on peut noter les suivantes :

La fonction	Sa signification
sin(x)	le sinus de x (en radian)
cos(x)	le cosinus de x (en radian)
tan(x)	le tangent de x (en radian)
asin(x)	l'arc sinus de x (en radian)
acos(x)	l'arc cosinus de x (en radian)
atan(x)	l'arc tangent de x (en radian)
sqrt(x)	la racine carrée de x ➔$\sqrt{x}$
abs(x)	la valeur absolue de x ➔ |x|
exp(x)	$= e^x$
log(x)	logarithme naturel de x ➔$ln(x)=log_e(x)$
log10(x)	logarithme à base 10 de x ➔ $\log_{10}(x)$
imag(x)	la partie imaginaire du nombre complexe x
real(x)	la partie réelle du nombre complexe x
round(x)	arrondi un nombre vers l'entier le plus proche
floor(x)	arrondi un nombre vers l'entier le plus petit ➔max{n|n≤x, n entier}
ceil(x)	arrondi un nombre vers l'entier le plus grand ➔min{n|n≥x, n entier}

MATLAB offre beaucoup de commandes pour l'interaction avec l'utilisateur. Nous nous contentons pour l'instant d'un petit ensemble, et nous exposons les autres au fur et à mesure de l'avancement du cours.

La commande	Sa signification
who	Affiche le nom des variables utilisées
whos	Affiche des informations sur les variables utilisées
clear x y	Supprime les variables x et y
clear, clear all	Supprime toutes les variables
clc	Efface l'écran des commandes
exit, quit	Fermer l'environnement MATLAB
format	Définit le format de sortie pour les valeurs numériques format long : affiche les nombres avec 14 chiffres après la virgule format short: affiche les nombres avec 04 chiffres après la virgule format bank : affiche les nombres avec 02 chiffres après la virgule format rat :affiche les nombres sous forme d'une ration (a/b)

1.2.4 La priorité des opérations dans une expression :

L'évaluation d'une expression s'exécute de gauche à droite en considérant la priorité des opérations indiquée dans le tableau suivant :

Les opérations	La priorité (1=max, 4=min)
Les parenthèses (et)	1
La puissance et le transposé ^ et '	2
La multiplication et la division * et /	3
L'addition et la soustraction + et -	4

Par exemple **5+2*3 = 11** et **2*3^2 = 18**

a) Exercice récapitulatif :

Créer une variable x et donnez-la la valeur 2, puis écrivez les expressions suivantes :

- $3X^3-2X^2+4X$
- $\frac{e^{1+x}}{1-\sqrt{2x}}$
- $|\sin^{-1}(2x)|$
- $\frac{\ln(x)}{2x^3}$ -1

b) La solution

```
>> x=2 ;
>> 3*x^3-2*x^2+4*x ;
>>exp(1+x)/(1-sqrt(2*x)) ;
>>abs(asin(2*x)) ;
>>log(x)/(2*x^3)-1 ;
```

1.3 EXERCICES

Exercice 01 :

Evaluer les expressions suivantes :

1. 4 +5+7
2. 3^{17}
3. $\sin(0.1\pi)$
4. (2-(3-(3+7(1-2(3-5)))))

Exercice 02 :

Définir la variable et calculer $x = \frac{\pi}{4}$ et ,puis $y_1 = \sin(x)$ et$y_2 = \cos(x)$,puis $z = \tan(x)$ à partir de y_1 et y_2 .

Définir la variable x= [π/6, π/4, π/3] et calculer $y_1 = \sin(x)$ et $y_2 = \cos(x)$, puis $z = \tan(x)$ en utilisant exclusivement les vecteurs y_1 et y_2 précédents.

Exercice 03 :

```
>>1/6
       ans =
              0.1667

>>format long

>>1/6
       ans =
              ……
>>format bank

>>1/6
       ans =
              ……
>>format short

>>1/6
       ans =
              ……

>> 7.8*3.7
       ans =
              …….

>>format rat

>> 7.8*3.7
       ans =
              …/…

>>……
ans =
…/…
```

CHAPITRE 2

LES VECTEURS ET LES MATRICES

2 Chapitre 02 : les vecteurs et les matrices

MATLAB est envisagé d'admettre aux mathématiciens, scientifiques et ingénieurs le fonctionnement facile les mécanismes de l'algèbre linéaire. Par conséquent, l'utilisation des vecteurs et des matrices est très intuitif et commode en MATLAB.

2.1 LES VECTEURS :

Un vecteur est une liste ordonnée d'éléments. Si les éléments sont arrangés horizontalement on dit que le vecteur est en ligne, par contre si les éléments sont arrangés verticalement on dit que c'est un vecteur colonne.

Pour créer un *vecteur ligne* il suffit d'écrire la liste de ses composants entre crochets **[**et**]** et de les séparés par des espaces ou des virgules comme suit :

```
>> V = [17, 18, 13,-7]                    % Création d'un vecteur ligne V
      V =
         17     18     13     -7

>> U = [5 -2 3]                           % Création d'un vecteur ligne U
      U =
         5     -2      3
```

Pour créer un *vecteur colonne* il est possible d'utiliser une des méthodes suivantes :

1. écrire les composants du vecteur entre crochets **[**et**]** et de les séparés par des points-virgules (**;**) comme suit :

```
>> U = [5; -2; 3]                         % Création d'un vecteur colonne U
      U =
             5
            -2
             3
```

2. écrire verticalement le vecteur :

```
>> U = [
5
-2
3
]
      U =
             5
            -2
             3
```

3. calculer le transposé d'un vecteur ligne :

```
>> U = [5 -2 3]'                          % Création d'un vecteur colonne U
      U =
             5
            -2
             3
```

Si les composants d'un vecteur **X** sont ordonnés avec des valeurs consécutives, nous pouvons le noter avec la notation suivante :

```
X = premier_élément : dernier_élément
```

(Les crochets sont facultatifs dans ce cas)

Par exemple :

```
>> X = 1:12                                %on peut aussi écrire colon(1,12)
      X =
            1     2     3     4     5     6     7     8     9    10    11    12
>> X = [1:8]
      X =
            1     2     3     4     5     6     7     8     9    10    11    12
```

Si les composants d'un vecteur **X** sont ordonnés avec des valeurs consécutives mais avec un pas (d'incrémentation/décrémentation) différente de 1, nous pouvons spécifier le pas avec la notation :

```
X = premier_élément : le_pas : dernier_élément
```

(Les crochets sont facultatifs)

Par exemple :

```
>> X = [0:2:10]                    %le vecteur X contient les nombres pairs < 12
      X =
            0     2     4     6     8    10
>> X = [-4:2:6]                    %on peut aussi écrire colon(-4,2,6)
      X =
           -4    -2     0     2     4     6
>> X = 0:0.2:1                     %on peut aussi écrire colon(0,0.2,1)
      X =
      0    0.2000    0.4000    0.6000    0.8000    1.0000
```

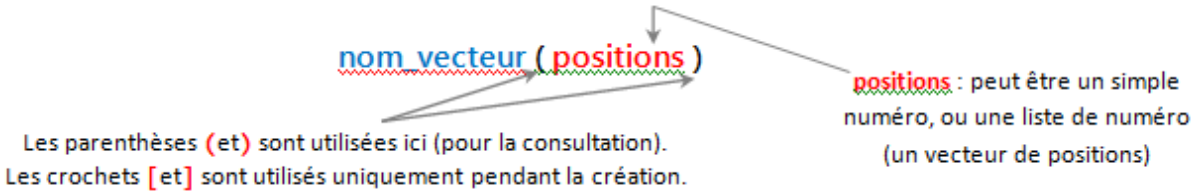
On peut écrire des expressions plus complexes comme :

```
>> V = [1:2:5,-2:2:1]
      V =
            1     3     5    -2     0
>> A = [1 2 3]
      A =
            1     2     3
>> B = [A, 4, 5, 6]
      B =
```

```
    1     2     3     4     5     6
```

2.1.1 Référencement et accès aux éléments d'un vecteur :

L'accès aux éléments d'un vecteur se fait en utilisant la syntaxe générale suivante :



Exemples 01:

```
>> V = [10, -17, 8,-5, 12]% création du vecteur V qui contient 5 éléments
V =
       10    -17     8    -5     12

>> V(4)                          %la 4eme position
       ans =
             -5

>>V(2:4)                         %de la deuxième position jusqu'au quatrième
       ans =
             -17     8    -5

>>V(5:-2:1)                      %de la 5eme posistion jusqu'à la 1ere avec le pas = -2
       ans =
              12     8     10

>>V(3:end)                       %de la 3eme position jusqu'à la dernière
       ans =
              8     -5     12

>>V([1,3,4])                     %la 1ere, la 3eme et la 5eme position uniquement
       ans =
             10     8    -5

>> V(1)=8                        %donner la valeur 8 au premier élément
       V =
             8    -17     8    -5     12

>> V(6)=-3                       %ajouter un sixième élément avec la valeur -3
       V =
             8    -17     8    -5     12    -3

>> V(9)=5                        %ajouter un neuvième élément avec la valeur 5
       V =
             8    -17     8    -5     12    -3     0     0     5

>> V(2)=[]                       %Supprimer le deuxième élément
```

```
      V =
            8     8     -5      12     -3      0      0      5
```

```
>>V(3:5)=[]                          % Supprimer du 3eme jusqu'au 5eme élément
      V =
            8     8      0      0      5
```

Exemple 02 :

```
>> V = [1:1:5]*2                     % création du vecteur V qui contient 5 éléments
V =
      2    4     6     8     10
```

```
>> V(3)                              %la 3eme position
      ans =
            6
```

```
>>V(2:4)                             %de la deuxième position jusqu'au quatrième
      ans =
            4     6     8
```

```
>>V(2:end)                           %de la 2eme position jusqu'à la dernière
      ans =
            4     6      8     10
```

```
>>V([5,3,1])                         %la 5eme, la 3eme et la 4ere position uniquement
      ans =
            10     6      2
```

2.1.2 Les opérations élément-par-élément pour les vecteurs :

Avec deux vecteurs $\vec{u}$ et$\vec{v}$, il est possible de réaliser des calcules élément par élément en utilisant les opérations suivantes :

L'opération	Signification	Exemple avec :
		`>> u = [-2, 6, 1] ;`
		`>> v = [ 3, -1, 4] ;`

+	Addition des vecteurs	`>> u+2`
		`ans =`
		`0 8 3`
		`>> u+v`
		`ans =`
		`1 5 5`

-	Soustraction des vecteurs	`>> u-2`
		`ans =`
		`-4 4 -1`
		`>> u-v`
		`ans =`
		`-5 7 -3`

.*	Multiplication élément par élément	`>> u*2`
		`ans =`
		`-4 12 2`
		`>> u.*2`

		ans = -4 12 2 **>> u.*v** ans = -6 -6 4
./	Division élément par élément	**>> u/2** ans = -1.0000 3.0000 0.5000 **>> u./2** ans = -1.0000 3.0000 0.5000 **>> u./v** ans = -0.6667 -6.0000 0.2500
.^	Puissance élément par élément	**>> u.^2** ans = 4 36 1 **>> u.^v** ans = -8.0000 0.1667 1.0000

L'écriture d'une expression tel que : u^2 génère une erreur car cette expression réfère a une multiplication de matrices (u*u doit être réécrite u*u' ou u'*u pour être valide).

Exemple 01 :

```
>> V = [1, 2, 3, 4]; W=[2, 4, 6, 8] ;
>> Z= V+W   %addition
Z =
   3    6    9    12
>>T=V-W      %soustraction
T =
   -1   -2   -3   -4
>>K=V*W'     %produit scalaire
K =
   60
>>S=V.*W     %produit terme-à-terme
S =
       28    18   32

>> V./W            % division terme-à-terme
ans =
       0.5000    0.5000     0.5000    0.5000

>> V+3                    %addition d'un scalaire
       ans =
              4    5    6    7

>> V-3                    %soustraction d'un scalaire
       ans =
              -2   -10    1
```

```
>> V*3                    %produit avec un scalaire
      ans =
            3    6    9    12
>> V/3                 %division sur un scalaire
      ans =
          0.3333      0.6667      1.0000    1.3333
>> V.^2                %mise à une puissance
      ans =
            149    16
```

Exemple 02 :

```
>>x=[1 2 3]; y=[4 5 6 7]; z=[8  9] ;
>> V=[x y z] %concaténation
      ans =
            1   2   3   4   5   6   7   8   9

>>length(V)  %taille
      ans =
            9
```

Exemple 03 :

```
>>a=[1 2]; b=[1;2];

>>length(a)  %taille
      ans =
            2
>>length(b)  %taille
      ans =
            2

>>size(a)    %nombre de lignes et de colonnes
      ans =
            1          2
>>size(b)    %nombre de lignes et de colonnes
      ans =
            2          1
```

Exemple 04 :

```
>>x=[2 4 5 6 7];
>>max(x)     %maximum
      ans =
            7
>>min(x)     %minimum
      ans =
            2
>>sum(x)     %somme des valeurs
      ans =
            24
```

```
>>prod(x)    %produit des valeurs
      ans =
            1680
>>mean(x)    %moyenne des valeurs
      ans =
            4.8000
>>median(x)  %la valeur médiane (après le tri)
      ans =
            5
>>var(x)     %la variance des valeurs
      ans =
            3.7000
>>std(x)     %l'écart type des valeurs
      ans =
            1.9235
```

$$sum(x) = \sum_{i=1}^{n} x_i$$

$$prod(x) = \prod_{i=1}^{n} x_i$$

$$mean(x) = \bar{x} = \frac{sum(x)}{n}$$

$$median(x) = \begin{cases} x(p+1) & si\ n\ =\ 2p+1\ (impair) \\ \dfrac{x(p)+x(p+1)}{2} & si\ n\ =\ 2p\ (pair) \end{cases}$$

$$var(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

$$std(x) = \sqrt{var(x)}$$

2.1.3 La fonction linspace :

La création d'un vecteur dont les composants sont ordonnés par intervalle régulier et avec un nombre d'éléments bien déterminé peut se réaliser avec la fonction :

*linspace(**début**, **fin**, nombre d'éléments).*

Le pas d'incrémentation est calculé automatiquement par Matlab selon la formule :

$$le\ pas = \frac{fin - debut}{nombre\ d'éléments - 1}$$

La fonction linspace(i, j, k) génère un vecteur de k éléments allant de i à j :

La taille d'un vecteur (le nombre de ses composants) peut être obtenue avec la fonction *length*comme suit :

```
>>length(X)          %la taille du vecteur X
      ans =
            4
```

Exemple 01:

```
>> X=linspace(1,10,5)            %un vecteur de cinq élément de 1 à 10
      X =
            1.0000     3.2500     5.5000    7.7500    10.0000

>> Y = linspace(15,60,6)          %un vecteur de 6 élément de 15 à 60
      Y =
            15    24    33    42    51    60
```

Exemple 02:

```
>>a = linspace(1,10,10)           %un vecteur de 10 élément de 1 à 10
      a =
            1    2    3    4    5    6    7    8    9    10
```

Exemple 03:

```
>>h = linspace(1,10,6)            %un vecteur de 6 élément de 1 à 10
      h =
            1.0000    2.8000    4.6000    6.4000    8.200     10.0000
```

- La fonction logspace(i, j, k) génère un vecteur k éléments allant de 10^i et 10^j :

```
>>h = logspace(0,2,5)             %un vecteur de 5 élément de 10^0 à 10^2
% 10^0   10^0.5   10^1   10^1.5   10^2
      h =
            1.0000    3.1623    10.0000    31.6228   100.0000
```

2.2 LES MATRICES

Une matrice est un tableau rectangulaire d'éléments (bidimensionnels). Les vecteurs sont des matrices avec une seule ligne ou une seule colonne (monodimensionnels).

Pour insérer une matrice, il faut respecter les règles suivantes :

- Les éléments doivent être mises entre des crochets **[** et**]**
- Les espaces ou les virgules sont utilisés pour séparer les éléments dans la même ligne
- Un point-virgule (ou la touche ***entrer***) est utilisé pour séparer les lignes

Pour illustrer cela, considérant la matrice suivante :

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Cette matrice pout être écrite en Matlab avec une des syntaxes suivantes :

```
>> A = [1,2,3,4;5,6,7,8;9,10,11,12] ;
```

```
>> A = [1 2 3 4;5 6 7 8;9 10 11 12] ;
```

```
>> A = [1,2,3,4
5,6,7,8
9,10,11,12] ;
>> A=[[1;5;9] , [2;6;10] , [3;7;11] , [4;8;12]] ;
```

Le nombre d'éléments dans chaque ligne (nombre de colonnes) doit être identique dans toutes les lignes de la matrice, sinon une erreur sera signalée par MATLAB. Par exemple :

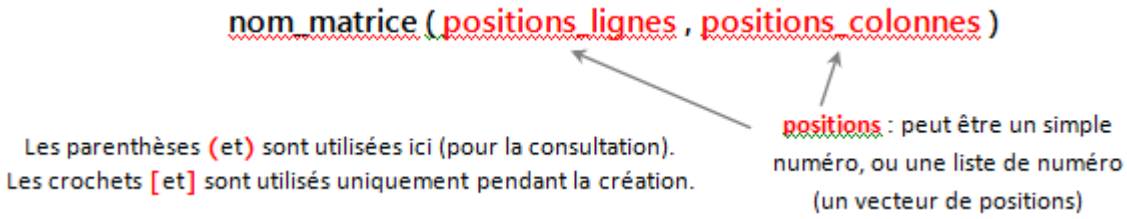
```
>> X = [1 2 ; 4 5 6]
      Error using vertcat
      CAT arguments dimensions are not consistent.
```

Une matrice peut être générée par des vecteurs comme le montre les exemples suivants :

```
>>x = 1:4                                %création d'un vecteur x
      x =
            1     2     3     4
>> y = 5:5:20                           %création d'un vecteur y
      y =
            5    10    15    20
>> z = 4:4:16                           %création d'un vecteur z
      z =
            4     8    12    16

>> A = [x ; y ; z]                      %A est formée par les vecteurs lignes x, y et z
      A =
            1     2     3     4
            5    10    15    20
            4     8    12    16

>> B = [x' y' z']                       %B est formée par les vecteurs colonnes x, y et z
      B =
            1     5     4
            2    10     8
            3    15    12
            4    20    16

>>C = [x ; x]                           %C est formée par le même vecteur ligne x 2 fois
      C =
            1     2     3     4
            1     2     3     4
```

2.2.1 Référencement et accès aux éléments d'une matrice:

L'accès aux éléments d'une matrice se fait en utilisant la syntaxe générale suivante :



Il est utile de noter les possibilités suivantes :

- L'accès à un élément de la ligne ***i*** et la colonne ***j*** se fait par : ***A(i,j)***
- L'accès à toute la ligne numéro ***i*** se fait par : ***A(i,:)***
- L'accès à toute la colonne numéro ***j*** se fait par : ***A(:,j)***

Exemples 01:

```
>> A = [1,2,3,4 ; 5,6,7,8 ; 9,10,11,12]      %création de la matrice A
      A =
            1     2     3     4
            5     6     7     8
            9    10    11    12

>> A(2,3)                          % l'élément sur la 2ème ligne à la 3ème colonne
      ans =
            7

>>A(1,:)                           %tous les éléments de la 1ère ligne
      ans =
            1     2     3     4

>>A(:,2)                           %tous les éléments de la 2ème colonne
      ans =
            2
            6
            10

>>A(2:3,:)                         %tous les éléments de la 2ème et la 3ème ligne
      ans =
            5     6     7     8
            9    10    11    12

>>A(1:2,3:4)                 %La sous matrice supérieure droite de taille 2x2
      ans =
            3     4
            7     8

>>A([1,3],[2,4])                   %la sous matrice : lignes(1,3) et colonnes (2,4)
      ans =
            2     4
            10    12
```

```
>>A(:,3)=[]                %Supprimer la troisième colonne
      A =
            1     2     4
            5     6     8
            9    10    12

>>A(2,:)=[]                %Supprimer la deuxième ligne
      A =
            1     2     4
            9    10    12

>> A=[A,[0;0]]                    %Ajouter une nouvelle colonne {ou A(:,4)=[0;0]}
      A =
            1     2     4     0
            9    10    12     0

>> A=[A;[1,1,1,1]]          %Ajouter une nouvelle ligne {ou A(3,:)=[1,1,1,1]}
      A =
            1     2     4     0
            9    10    12     0
            1     1     1     1
```

Les dimensions d'une matrice peuvent être acquises en utilisant la fonction ***size***. Cependant, avec une matrice A de dimension $m \times n$le résultat de cette fonction est un vecteur de deux composants, une pour ***m***et l'autre pour ***n***.

```
>> d = size(A)
      d =
            3     4
```

Ici, la variable ***d*** contient les dimensions de la matrice A sous forme d'un vecteur. Pour obtenir les dimensions séparément on peut utiliser la syntaxe :

```
>> d1 = size (A, 1)              %d1 contient le nombre de ligne (m)
      d1 =
            3
>> d2 = size (A, 2)              %d2 contient le nombre de colonne (n)
      d2 =
            4
```

2.2.2 Génération automatique des matrices

En MATLAB, il existe des fonctions qui permettent de générer automatiquement des matrices particulières. Dans le tableau suivant nous présentons-les plus utilisées :

La fonction	Signification

zeros(n)	Génère une matrice $n \times n$ avec tous les éléments = 0
zeros(m,n)	Génère une matrice $m \times n$ avec tous les éléments = 0
ones(n)	Génère une matrice $n \times n$ avec tous les éléments = 1
ones(m,n)	Génère une matrice $m \times n$ avec tous les éléments = 1
eye(n)	Génère une matrice identité de dimension $n \times n$
magic(n)	Génère une matrice magique de dimension $n \times n$
rand(m,n)	Génère une matrice de dimension m $\times$ n de valeurs aléatoires

- Les matrices suivent la même syntaxe que les vecteurs.
- Les composantes des lignes sont séparées par des virgules ou des espaces et chaque ligne est séparée de l'autre par un point virgule.
- La commande repmat(val, lin, col) crée une matrice pleine des valeurs dupliqués .
- Les commandes zeros(lin, col) et ones(lin, col) créent des matrices des zéros et des uns, respectivement.

Exemple :

```
>> A = [1 2 3; 4 5 6; 7 8 9]    %création de la matrice A
      A =
            1     2     3
            4     5     6
            7     8     9
>>c = repmat(1, 4, 3)     %création de la matrice c, 4 lignes et 3 colonnes
      A =
            1     1     1
            1     1     1
            1     1     1
            1     1     1
>> A=ones(2,5), B=zeros(4,4)
      A =
            1     1     1     1     1
            1     1     1     1     1
      B =
            0     0     0     0
            0     0     0     0
            0     0     0     0
            0     0     0     0
```

- Comme pour les vecteurs, une matrice peut être définie aléatoirement en utilisant les commande rand(lin, col) et randi(max, lin, col) :

```
>> z= rand(3,3)     %des valeurs aléatoires entre 0 et 1
z=
       0.9575       0.9706       0.8003
```

```
    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
```

```
>>p= randi(10,3,3)     %des valeurs entiers aléatoires entre 1 et 10
p=

    10     7    10
     8     1     7
    10     9     8
```

- La matrice identité peut être obtenue par la commande eye(lin, col) :

```
>>eye(3,3)     %matrice identité de 3X3
ans=

     1     0     0
     0     1     0
     0     0     1
```

```
>>eye(4,4)     %matrice identité de 4X4
ans=

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

```
>>eye(4,3)     %matrice identité de 4X3
ans=

     1     0     0
     0     1     0
     0     0     1
     0     0     0
```

- Une matrice peut être produise à partir d'une autre matrice ou un autre vecteur en utilisant la commande reshape(Vec, [lin, col]) en changeant sa dimension (le nombre d'éléments doit être identique):

```
>>v = [1 2 3 4 5 6 7 8 9 10 11 12]     %vecteur ligne de 12 éléments
    v =
     1     2     3     4     5     6     7     8     9    10    11    12
>> A=reshape(v,3,4)     %matrice de 3X4
A=
     1     4     7    10
     2     5     8    11
     3     6     9    12
```

```
>> B=reshape(v,4,3)      %matrice de 4X3
B=

     1     5     9
     2     6    10
     3     7    11
     4     8    12

>> C=reshape(v,2,6)      %matrice de 2X6
C=
     1     3     5     7     9    11
     2     4     6     8    10    12

>> D=reshape(v,6,2)      %matrice de 6X2
D=
     1     7
     2     8
     3     9
     4    10
     5    11
     6    12
```

2.2.3 Les opérations de base sur les matrices

L'opération	Signification
+	L'addition
-	La soustraction
.*	La multiplication élément par élément
./	La division élément par élément
.\	La division inverse élément par élément
.^	La puissance élément par élément
*	La multiplication matricielle
/	La division matricielle (A/B) = ($A*B^{-1}$)

Les opérations élément par éléments sur les matrices sont les mêmes que ceux pour les vecteurs (la seule condition nécessaire pour faire une opération élément par élément est que les deux matrices aient les mêmes dimensions). Par contre la multiplication ou la division des matrices requiert quelques contraintes (consulter un cours sur l'algèbre matricielle pour plus de détail).

Exemple :

```
>> A=ones(2,3)
     A =
          1     1     1
          1     1     1
>> B=zeros(3,2)
     B =
```

```
     0     0
     0     0
     0     0
>> B=B+3
   B =
     3     3
     3     3
     3     3
>> A*B
   ans =
     9     9
     9     9
>> B=[B , [3 3 3]']          %ou bien B(:,3)=[3 3 3]'
   B =
     3     3     3
     3     3     3
     3     3     3
>> B=B(1:2,:)                %ou bien B(3,:)=[]
   B =
     3     3     3
     3     3     3
>> A=A*2
   A =
     2     2     2
     2     2     2
>> A.*B
   ans =
     6     6     6
     6     6     6
>> A*eye(3)
   ans =
     2     2     2
     2     2     2
```

2.2.4 Fonction utiles pour le traitement des matrices

Voici quelques fonctions parmi les plus utilisées concernant les matrices:

La fonction	L'utilité	Exemple d'utilisation

det	Calcule de déterminant d'une matrice	`>> A = [1,2;3,4] ;`
		`>> det(A)`
		`ans =`
		`-2`

inv	Calcule l'inverse d'une matrice	`>> inv(A)`
		`ans =`
		`-2.0000  1.0000`
		`1.5000  -0.5000`

rank	Calcule le rang d'une matrice	`>> rank(A)`
		`ans =`
		`2`

trace	Calcule la trace d'une matrice	`>> trace(A)`
		`ans =`
		`5`

eig	Calcule les valeurs propres	**>> eig(A)** **ans =** **-0.3723** **5.3723**
dot	Calcule le produit scalaire de 2 vecteurs	**>> v = [-1,5,3];** **>> u = [2,-2,1];** **>> dot(u,v)** **ans =** **-9**
norm	Calcule la norme d'un vecteur	**>> norm(u)** **ans =** **3**
cross	Calcule le produit vectoriel de 2 vecteurs	**>>cross(u,v)** **ans =** **-11 -7 8**
diag	Renvoie le diagonal d'une matrice	**>> diag(A)** **ans =** **1** **4**
diag(V)	Crée une matrice ayant le vecteur V dans le diagonal et 0 ailleurs.	**>> V = [-5,1,3]** **>> diag(V)** **ans =** **-5 0 0** **0 1 0** **0 0 3**
tril	Renvoie la partie triangulaire inferieure	**>> B=[1,2,3;4,5,6;7,8,9]** **B =** **1 2 3** **4 5 6** **7 8 9** **>> tril(B)** **ans =** **1 0 0** **4 5 0** **7 8 9** **>>tril(B,-1)** **ans =** **0 0 0** **4 0 0** **7 8 0** **>>tril(B,-2)** **ans =** **0 0 0** **0 0 0** **7 0 0**

triu	Renvoie la partie triangulaire supérieure	**>> triu(B)** **ans =** **1 2 3** **0 5 6** **0 0 9** **>>triu(B,-1)** **ans =** **1 2 3** **4 5 6** **0 8 9** **>>triu(B,1)** **ans =** **0 2 3** **0 0 6** **0 0 0**

- MATLAB fournie quelque opérations pour extraire les propriétés de base des matrices :

```
>> A = [1 2 1; 1 1 2; 2 21]      %création de la matrice A
      A =
            1     2     1
            1     1     2
            2     2     1
>>det(A)     %déterminant
      ans =
            3
>>size(A)    %taille :lignes,colonnes
      ans =
            3     3
>>diag(A)    %le vecteur du diagonal
      ans =
            1
            1
            1
>>trace(A)   %la trace
      ans =
            3

>>triu(A)    %matrice triangulaire superieure
      ans =
            1     2     1
            0     1     2
            0     0     1

>>tril(A)    %matrice triangulaire inférieure
      ans =
            1     0     0
            1     1     0
            2     2     1

>>A'   %le transposé
      ans =
```

```
            1     1     2
            2     1     2
            1     2     1
```

```
>>A_1=inv(A) %l'inverse
      A_1 =
            -1.0000           0       1.0000
             1.0000      -0.3333    -0.3333
                  0       0.6667      -0.3333
```

```
>>A*A_1
      ans =
             1.0000           0       -0.0000
             1.0000      1.0000             0
                  0           0        1.0000
```

2.2.5 Rang d'une matrice et forme échelonnée

❖ Une matrice A est dite échelonnée en lignes si :

- Chaque ligne non nulle de A commence avec strictement plus de 0que la ligne précédente.
- Les lignes nulles (ne contenant que des 0) de A viennent en bas après les lignes non nulles.

❖ Une matrice A est dite en forme échelonnée réduite si :

- A est échelonnée.
- Pour chaque ligne, la première valeur non-nulle est la seule non-nulle dans sa colonne

❖ Le rang d'une matrice A est le nombre de lignes non nulles dans sa forme échelonnée en lignes.

❖ La commande rank retourne le rang d'une matrice.

```
>> A = [1 2 3; 1 3 2; 2 32]     %matrice singulière
      A =
            1     2     3
            1     3     2
            2     3     2
>>det(A)     %déterminant
      ans =
            -5
>>rank(A)    %rang
      ans =
            3
>>B = [1 2 3; 2 4 6; 2 3 2]     %matrice non-singulière
```

```
      B =
            1     2     3
            2     4     6
            2     3     2

>>det(B)     %déterminant
      ans =
            0
>>rank(B)    %rang
      ans =
            2
>>C = [1 2 3; 2 4 6; 3 6 9]     %matrice non-singulière
      C =
            1     2     3
            2     4     6
            3     6     9
>>det(C)     %déterminant
      ans =
            0
>>rank(C)    %rang
      ans =
            1
```

2.2.6 Opération sur les matrices

❖ En algèbre, Il y a plusieurs opérations de base qui s'effectuent sue les matrices. Néanmoins, ces opérations souvent exigent des pré-conditions avant s'exécuter :

a) A + B, A - B : (addition et soustraction) A et B ont la même dimension, sauf pour les scalaires.

b) A.*B, A./B : (multiplication et division terme à terme) A et B ont la même dimension, sauf pour les scalaires.

c) A*B : (multiplication) le nombre des colonnes de A égale au nombre de lignes de B, sauf pour les scalaires.

d) A/B : (division) A et B sont carrées et de même dimension, et B est singulière (inversible, ou déterminant non nulle) , sauf pour les scalaires.

```
>> A = [1 2 3; 4 5 6; 7 8 9]     %matrice A
      A =
            1     2     3
            4     5     6
            7     8     9
>> B=A+2                  %matrice B
      B =
            3     4     5
            6     7     8
```

```
             9     10     11
>> C=A-2                    %matrice C
      B =
            -1     0     1
             2     3     4
             5     6     7
>> D=A*2                    %matrice D
      D =
             2     4     6
             8    10    12
            14    16    18
>> E=A^2                    %matrice A*A
      D =
            30       36     42
            66       81     96
           102      126    150
>> F=2.^A                   %mettre chaque élément comme une puissance
      F =
             2       4       8
            16      32      64
           128     256     512
>> J=A/2
      J =
           0.5000       1.0000     1.5000
           2.0000       2.5000     3.0000
           3.5000       4.0000     4.5000
>> K = [1 1 2; 1 1 1; 2 1 1]    %matrice K
      k =
            1     1     2
            1     1     1
            2     1     1
>>A+k
      ans =
            2     3     5
            5     6     7
            9     9    10
>>A-k
      ans =
            0     1     1
            3     4     5
            5     7     8
>>A*k
      ans =
             9      6      7
            21     15     19
            33     24     31
>>A/k
      ans =
             1      2     -1
             1      5     -1
```

```
             1      8      -1
>>A./k
      ans =
             1.0000      2.0000      1.5000
             4.0000      5.0000      6.0000
             3.5000      8.0000      9.0000
```

Comme pour les vecteurs, la commande find(A) trouve les valeurs non nulles dans une matrice. find(cond A) trouve les valeurs satisfaisantes de la condition en question.

```
>>A = [1 0 2; 4 5 0; 7 0 8]      %matrice A
      A =
            1     0     2
            4     5     0
            7     0     8
>>[lin col val]=find(A)   %trouver les valeurs non nulles
      lin =
            1
            2
            3
            2
            1
            3
col =
            1
            1
            1
            2
            3
            3
val =
            1
            4
            7
            5
            2
            8


>>[lin col val]=find(A>5) %trouver les valeurs supérieure à 5
      lin =
            3
            3
col =
            1
            3
val =
            1
            1
```

```
>>ind=sub2ind (size(A), lin, col)%convertir les indices lin, col en indice linéaires
      ind =
            3
            9
>>A(ind)                          %Les valeurs >5

ans =
            7
            8
```

2.2.7 L'indexage dans une matrice

La modification d'un élément (ou plusieurs éléments) dans une matrice peut être effectué par de son index comme suit: A(i), A(i, j): l'élément de l'index i ou (i, j) de la matrice A.

```
>> A = [1 2 3; 4 5 6; 7 8 9]     %création de la matrice A
      A =
            1     2     3
            4     5     6
            7     8     9
>> A(7)                          %indexage linéaire
      ans =
            3
>>A(1, 3)                        %indexage ligne-colonne
      ans =
            3
>>index=sub2ind (size(A), 1, 3) %conversion de l'indexage ligne-colonne à l'indexage linéaire
      index =
            7
>>[i, j]=ind2sub (size(A), 7)    %conversion de l'indexage linéaire à l'indexage ligne-colonne
      i =
            1
      j =
            3
```

A(i: j), A(i1: i2, j1: j2): une sous-matrice de A à partir de l'index linéaire i jusqu'à l'index j (i1 et i2 index des lignes, j1 et j2 index des colonnes).

A(i: pas: j), A(i1: pas: i2, j1: pas: j2): une sous-matrice de A à partir de l'index i jusqu'à l'index j avec un pas (i1 i2 index des lignes, j1 j2 index des colonnes).

A([i1 i2 … in]), A([i1 i2 … in] ,[j1 j2 … jn]): une sous-matrice de A en prendre en considération les indices i1, i2, …, in (pour l'indexage ligne-colonne, on considère i1,i2,…,in des indices des lignes, j1,j2,…,jn indices des colonnes).

```
>> A = [1 2 3; 4 5 6; 7 8 9]    %création de la matrice A
      A =
            1     2     3
            4     5     6
            7     8     9
>>A(1:5)
      ans =
            1     4     7     2     5
>>A(1:2 5)
      ans =
            1     7     5
>>A(1:2, 1:2)
      ans =
            1     2
            4     5
>>A(1:2:3, 1:2 :3)
      ans =
            1     3
            7     9
>>A([1 5 6 8])
      ans =
            1     5     8     6
>>A([1 3], [1 2])
      ans =
            1     2
            7     8
>>A(1, :)
      ans =
            1     2     3
>>A( :, 2)
      ans =
            2
            5
            8
>>A( :, :)
      ans =
            1     2     3
            4     5     6
            7     8     9
>>A( :)
      ans =
           1
           2
           3
           4
           5
           6
           7
           8
           9
```

2.3 EXERCICES

Exercice 01 :

Soit les matrices : $\boldsymbol{A} = \begin{pmatrix} -1 & 1 \\ 3 & 0 \\ 0 & 2 \end{pmatrix}$ $\boldsymbol{B} = \begin{pmatrix} \sqrt{2} & -2 \\ 0.8 & 1.2 \end{pmatrix}$

1) Peut-on calculez les expressions suivantes :
   - A^2 ?
   - A.^2 ?
   - A/2 ?
   - A+zeros(3,3)?
   - A.*ones(3,2) ?
   - A*B ?
   - A*eye(2) ?
   - A*eye(3) ?
2) Calculez les expressions suivantes :
   - ceil(B)-floor(B)
   - B = floor(B)
   - [A , A]'
   - A.*(2*ones(3,2))
   - A+(2+zeros(3,2))
   - A == [-1 , 1 ; 2, 4 ; 0, 5]
   - isequal(A, [-1 , 1 ; 2, 4 ; 0, 5])
   - [1 , 2 ; 5 , 6] > [2 , 1 ; 4 ,7]

Exercice 02 :

Soit la matrice : M = [5 :-1 :0 ; 2 :2 :12 ; 11 :-3 :-4]
Calculez les opérations suivantes :

- M(1 : 2 , 2 : 6)
- M(2 : 3 , :)
- M(2 , [2 , 4 , 6])
- V = M(1 , end : -1 : 1)
- V(end-1)
- V(end-4)
- M(end :-1 : 1 , end : -3 : 1)

Exercice 03 :

Compléter les phrases suivantes avec les mots indiqués en bas:

1. La fonction ..... arrondi un nombre vers l'entier le plus petit.
2. La commande ..... ferme l'environnement Matlab.
3. Pour obtenir les dimensions d'une matrice, on utilise la fonction .....
4. Pour obtenir la taille d'un vecteur, on utilise la fonction .....
5. La fonction ..... (m,n) génère une matrice de dimension m×n de valeurs aléatoires.
6. Le symbole **%** est utilisé pour définir des ..... en Matlab.
7. la fonction ..... est utilisée pour lire une valeur donnée par l'utilisateur, et la fonction ..... pour afficher des informations.

Les mots sont :

length, input, exit, commentaires, size, floor, rand, disp

Exercice 04:

Soit les deux matrices : $\boldsymbol{A} = \begin{pmatrix} 1 & 0 \\ -2 & 1 \\ 0 & 5 \end{pmatrix}$, $\boldsymbol{B} = \begin{pmatrix} -1 & 0 & -3 \\ 2 & 1 & 0 \end{pmatrix}$

Calculez les expressions suivantes :

```
>> A+ones(3,2)
   ans =        ...      ...
                ...      ...
                ...      ...
>> 3+zeros(2)-1
   ans =        ...      ...
                ...      ...
>> A*B
   ans =        ...      ...      ...
                ...      ...      ...
                ...      ...      ...
>> B'*eye(2)
   ans =        ...      ...
                ...      ...
                ...      ...
>> A(1:2 , : ) + B( : , 1:2)
   ans =        ...      ...
                ...      ...
```

Est-ce que les expressions suivantes sont envisageables (répondez par **oui** ou **non** uniquement) ?

1) `A+B`

Oui ☐ Non ☐

2) `A.*B`

Oui ☐ Non ☐

3) `A*eye(3)`

Oui ☐ Non ☐

4) `A'+B`

Oui ☐ Non ☐

5) `B*A`

Oui ☐ Non ☐

Exercice 05:

Soit les deux matrices : $\boldsymbol{A} = \begin{pmatrix} 0 & 3 \\ -1 & 1 \end{pmatrix}$, $\boldsymbol{B} = \begin{pmatrix} 2 & 1 \\ 2 & 0 \end{pmatrix}$

Calculez les expressions suivantes :

- `A.*B`
- `A.^B`
- `B+ones(2)`
- `A*eye(2)`
- `A+(B(2,1)+zeros(2,2))`

Exercice 06:

Soit les trois matrices : $\mathbf{A} = \begin{pmatrix} 2 & 0 & 1 \\ -1 & -3 & 2 \end{pmatrix}$, $\mathbf{B} = \begin{pmatrix} -1 & 1 \\ 4 & 3 \end{pmatrix}$, $\mathbf{C} = \begin{pmatrix} 1 & 6 \\ 0 & -2 \\ 4 & 0 \end{pmatrix}$

Quel est le résultat d'évaluation des expressions suivantes :

1) `A*C`
2) `A.*(C')-[B , ones(2,1)]`
3) `B.^ones(2) + B.^zeros(2)`
4) `C + [B ; [-4 1]]`
5) `D = [A ; C']; D(:,2)=[]`
6) `D = D(end:-2:1 , :)`
7) `A(:,2) = [3 ; 5]`

Exercice 07:

Confirmez ou infirmez par (**oui**) ou (**non**) chacune des assertions suivantes :

	L'énoncé	La réponse
1)	Pour afficher les résultats des calculs sous forme de fractions on utilise la commande : **`format rat`**	.....
2)	Pour une matrice carrée A, les deux expressions (**`A^2`**) et (**`A.^2`**) donnent toujours le même résultat	.....
3)	Les deux expressions **`diag(ones(1,50))`** et **`eye(50)`** donnent le même résultat	.....
4)	Pour comparer deux matrices et tester leurs égalités on peut : • soit utiliser la fonction **isequal** • soit utiliser le symbole de comparaison == Mais ces deux méthodes ne donnent pas le même type de résultat	.....
5)	La commande **`hold on`** active le mode préservation des graphiques ce qui permet de dessiner plusieurs courbes dans la même figure	.....
6)	Pour tout nombre réel R, les expressions **`round(R)`**, **`floor(R)`** et **`ceil(R)`** donnent toujours le même nombre entier	.....
7)	Le nom **Matlab** est l'abréviation de **MAT**rix **LAB**oratory	.....

Exercice 08:

Soit les trois matrices : $\mathbf{A} = (\ 1 \quad 3 \quad -2 \quad 0)$, $\mathbf{B} = \begin{pmatrix} 0 & -1 \\ -2 & 3 \end{pmatrix}$, $\mathbf{C} = \begin{pmatrix} 1 & 5 & 6 \\ 7 & -3 & 2 \end{pmatrix}$

Quel est le résultat Matlab pour les expressions suivantes (avec éclaircissement) :

1) `X = [A ; A ; A]`
2) `X(: , end:-2:1) = ones(3,2)`
3) `[B' B'].^ 2`
4) `1-B^2`
5) `B.*C(: , [1,3])`
6) `Z = B == zeros(2)`

Exercice 09:

Répondez par (**vrai**) ou (**faux**) sans justification pour chacune des hypothèses suivantes :

Bonne réponse (+01pt), *mauvaise réponse* (-0.5pt), *aucune réponse* (0pt)

1. Pour toute matrice carrée A, les deux expressions suivantes donnent toujours le même résultat : **`tril(triu(A))`** et **`triu(tril(A))`**
2. L'instruction
```
while 0
      disp('itération en cours... ')
end
```
est une boucle infinie.
3. On peut remplacer la fonction **`mod(a,b)`** qui calcule le reste de la division entière de deux nombre naturels **a** et **b** (le modulo) par la formule : **`a-floor(a/b)*b`**
4. L'instruction
```
for i = 5:2
      disp(i)
end
```
ne fait rien.
5. Pour toute matrice carrée **M** avec **n** lignes (`n>0`) l'expression suivante renvoie toujours **0** :

`isequal( diag(diag(M)) , M.*eye(n))`

Exercice 10:

Répondez par (**vrai**) ou (**faux**) :

	L'énoncé	La réponse	
		Vrai	Faux
1)	Le symbole **%** est utilisé en Matlab pour calculer le modulo.		
2)	La commande **`clear`** efface la fenêtre des commandes sans supprimer les variables.		
3)	L'instruction **`switch`** permet la réalisation des boucles iteratives.		
4)	Pour toute matrice **A** de dimension **n**x**m** l'expression **`(A == A)`** donne toujours la matrice **`ones(n,m)`** .		
5)	Pour toutes matrice **A** et B de dimension **n**x**m** l'expression **`(A>B + B>A)`** donne toujours la matrice **`ones(n,m)`** .		

Exercice 11:

Répondez par (**vrai**) ou (**faux**) sans justification pour chacune des hypothèses suivantes :

Bonne réponse (+01pt), *mauvaise réponse* (-0.5pt), *aucune réponse* (0pt)

1. Pour toute matrice carrée A, les deux expressions suivantes donnent toujours le même résultat : **`tril(triu(A))`** et **`triu(tril(A))`**
2. L'instruction
```
while 0
      disp('itération en cours... ')
end
```
est une boucle infinie.
3. On peut remplacer la fonction **`mod(a,b)`** qui calcule le reste de la division entière de deux nombre naturels **a** et **b** (le modulo) par la formule : **`a-floor(a/b)*b`**
4. L'instruction
```
for i = 5:2
      disp(i)
end
```
ne fait rien.
5. Pour toute matrice carrée **M** avec **n** lignes (**n>0**) l'expression suivante renvoie toujours **0** :

`isequal( diag(diag(M)) , M.*eye(n))`

6. Il est toujours possible de remplacer l'instruction **`for`** par l'instruction **`while`** mais le contraire peut parfois s'avérer irréalisable.

La solution Exercice 01 :

```
>> A = [-1,1;3,0;0,2]
A =
    -1     1
     3     0
     0     2
>> B = [sqrt(2),-2;0.8,1.2]
B =
    1.4142   -2.0000
    0.8000    1.2000
```

- A^2 ? non
- A.^2 ? oui
- A/2 ? oui
- A+zeros(3,3)? non
- A.*ones(3,2) ? oui
- A*B ? oui
- A*eye(2) ? oui
- A*eye(3) ? non

La solution Exercice 02 :

```
>> M = [5 :-1 :0 ; 2 :2 :12 ; 11 :-3 :-4]
M =
     5     4     3     2     1     0
     2     4     6     8    10    12
    11     8     5     2    -1    -4

>> M(1 : 2 , 2 : 6)
ans =
     4     3     2     1     0
     4     6     8    10    12


>> M(2 : 3 , : )
ans =
     2     4     6     8    10    12
    11     8     5     2    -1    -4

>> M(2 , [2 , 4 , 6])
ans =
     4     8    12

>> V = M(1 , end : -1 : 1)
V =
     0     1     2     3     4     5

>> V(end-1)
ans =
     4

>> V(end-4)
ans =
     1

>> M(end :-1 : 1 , end : -3 : 1)
ans =
    -4     5
    12     6
     0     3
```

CHAPITRE 3

FONCTIONS

3 Chapitre 03 : Fonctions

La notion de fonction sous MATLAB fait l'objet de ce chapitre. Les exemples proposés permettent de saisir les notions de fonction *inline*, anonymes et l'importance des fichiers .m qui constituent un élément important dans la programmation d'application sous MATLAB ou les fonctions jouent le rôle des fonctions et procédures des langages de programmation usuels.

3.1 INTRODUCTION :

Les fonctions sont des enchainements de commandes MATLAB regroupées sous un nom de fonction permettant de commander leur exécution. On peut mettre dans une fonction un groupe de commandes destiné à être exécuté plusieurs fois au cours du calcul avec éventuellement des valeurs de paramètres différentes.

Les premières fonctions que l'on rencontre dans MATLAB sont les fonctions internes, ou prédéfinies. Pour les utiliser, il faut connaitre la syntaxe standard de MATLAB. A titre d'exemple la fonction exponentielle s'écrit exp(x) et le logarithme néperien log(x) :

```
>>x=exp(1)
      x =
            2.7183
>>log(x)
      ans =
            1
```

Le logarithme décimal s'écrit log10(10).

```
>>log10(10)
      ans =
            1
```

On dispose également de fonctions permettant de manipuler des nombres complexes. Par exemple :

```
>> x=-2+5i
      x =
            -2.0000 + 5.0000i
>> a=real(x)
      a =
            -2
>> b=img(x)
```

```
      b =
            5

>> F=abs(x)
      F =
            5.3852
>>alpha=angle(x)
      alpha =
            1.9513
```

Un complexe est généralement représenté sous la forme algébrique ou cartésienne.

Un complexe peut également être représenté sous la forme polaire $\rho e^{\theta i}$, où ρ est son module et θ son argument.

Exemple :

```
>>C=-2+5i                          %forme algébrique ou cartésienne
      C =
            2.0000 + 2.0000i

>>theta=angle(C)           %forme algébrique ou cartésienne
      theta =
            0.7854
>>ro=abs(C)                         %module de C
      ro =
            2.8284
>>ro*exp(theta*i)                   %forme polaire
      ans =
            2.0000 + 2.0000i
>>[theta, ro]= cart2pol(2,2)        %conversion au polaire
      theta =
            0.7854
      ro =
            2.8284
>>[x, y]= pol2cart(theta,ro)        %conversion au cartésien
      x =
            2.0000
      y =
            2.0000
```

En général, les fonctions peuvent être :

- Des fonctions prédéfinies dans la bibliothèque de MATLAB,
- Des fonctions dites*inline*, définies par l'utilisateur dans le corps du programme,

- Des fonctions dites anonymes précédées par le symbole@, définies par l'utilisateur,
- Des fonctions enregistrées dans des fichiers externes indépendants appelés fichiers .m

3.2 FONCTIONS PRÉDÉFINIES

MATLAB contient une riche bibliothèque de fonctions prédéfinies. Pour les utiliser on doit connaitre leur nono L'aide en ligne dans la fenêtre de commande, en mode graphique ou à l'aide d'un moteur de recherche sur internet, permettent de se documenter sur une fonction donnée, Les fonctions les plus courantes sont résumées dans les tableaux ci-dessous.

Tableau 01 :Fonctions trigonométrique

Fonctions trigonométrique	Description
sin(x)	le sinus de x (en radian)
cos(x)	le cosinus de x (en radian)
tan(x)	le tangent de x (en radian)
asin(x)	l'arc sinus de x (en radian)
acos(x)	l'arc cosinus de x (en radian)
atan(x)	l'arc tangent de x (en radian) $-\pi/2 \leq \text{atan}(x) \leq \pi/2$
atan2(x,y)	*atan2(x,y)* : $-\pi/2 \leq \text{atan}(x,y) \leq \pi/2$
sinh(x)	*sinus hyperbolique*
conh(x)	*cosinus hyperbolique*
tanh(x)	*tangent hyperbolique*
asinh(x)	*arc sinus hyperbolique*
acosh(x)	*arc cosinus hyperbolique*
atanh(x)	*arc tangent hyperbolique*

Tableau 02 :Autres fonctions mathématiques courantes

Autres fonction	Sa signification
abs(x)	la valeur absolue de x ➔ |x|
exp(x)	Exponentielle de base « e » $=e^x$
angle(x)	Argument du complexe x
sqrt(x)	Racine carrée de x
log(x)	logarithme naturel de x ➔$ln(x)=log_e(x)$
log10(x)	logarithme à base 10 de x ➔ $\log_{10}(x)$
imag(x)	la partie imaginaire du nombre complexe x
real(x)	la partie réelle du nombre complexe x
conj(x)	Complexe conjugué de x
round(x)	arrondi un nombre vers l'entier le plus proche

floor(x)	arrondi un nombre vers l'entier le plus petit ➔max{n|n≤x, n entier}
fix(x)	Arrondi par défaut de x
ceil(x)	arrondi un nombre vers l'entier le plus grand ➔min{n|n≥x, n entier}
Rem(x,y)	Le reste de la division x/y

Dans le chapitre précédent nous avons déjà utilisé quelques fonctions relatives aux matrices, comme l'inverse (inv (A)), le déterminant (det(A)).

D'autres fonctions sont disponibles.

Tableau 03 :Principales opérations sur les matrices.

Commande	Sa signification
det (A)	Renvoie le déterminant de A : celle-ci doit être carrée.
trace(A)	Renvoie le trace de A.
rank(A)	Renvoie le rang de A (dimension de l'image de l'application associée à A).
diag(A)	Renvoie la première diagonale de A.
norm(A)	Renvoie la norme euclidienne de V ; V est un vecteur.
mean(A)	Renvoie une liste contenant la moyenne des éléments de chaque colonne.
sum(A)	Renvoie une liste contenant la somme des éléments de chaque colonne.
prod(A)	Renvoie une liste contenant le produit des éléments de chaque colonne.
max(A)	Renvoie une liste contenant la valeur maximale de chaque colonne.
min(A)	Renvoie une liste contenant la valeur minimale de chaque colonne.
length(A)	Renvoie le maximum entre le nombre de lignes et de colonnes ; si A est un vecteur, length (A) est le nombre d'élément dans le vecteur.

3.3 LES FONCTIONS inline

Une façon relativement simple pour définir une fonction sous MATLAB consiste à utiliser la commande *inline* qui permet d'écrire des expressions mathématiques dépendant d'un argument d'entrée. La syntaxe est :

Fonction = inline(' expression', 'var')

- Inline est mot réservé pour définir une fonction inline,
- fonction est le nom de la fonction,
- expression est l'expression de la fonction sous sa forme mathématique,

- var est le nom de la variable.

```
>> f=inline('x^2+x/5+1', 'x')
```

Si on veut calculer la valeur de cette fonction pour x=5 on écrit

```
>>f(x)
      ans =
            127
```

Pour pouvoir appliquer cette fonction à un vecteur il est nécessaire d'utiliser les opérateurs appropriés : .^ , et ./. Pour l'exemple précédentl'écriture général est

```
>> f=inline('x.^2+x/5+1', 'x')
```

En appliquant cette fonction au vecteur ligne x=[0 :0.3 :1] on trouve :

```
>> f=inline('x.^2+x/5+1', 'x')
>>f=
    1.0000   1.0870   1.3360   1.909
```

On peut également :

```
>>f(0:0.3:1)
```

La commande inline peut aussi être utilisée pour définir des fonctions qui dépendent de plusieurs variables :

```
>>f2=inline('x.^3.*y+x/5+1+y', 'x', 'y')
   Inline foncion

F2(x,y)= x.^3.*y+x/5+1+y'
```

```
>>f2(3,2)
      ans =
            57.6000
```

3.4 LES FONCTIONS 'anonymes'

On utilise également les fonctions dites anonymes définies par la commande @ . Cette fonction est dite anonyme car iln 'est pas nécessaire de la sauvegarder dans un fichier. La syntaxe est :

nom_de_ fonction=@(input_variable) (expression)

ou

- Le symbole @ :définit la fonction anonyme nomfonction.
- Input_variable : est le nom de la variable d'entrée.
- Expression : est l'expression mathématique de la fonction qui doit etre écrire sur une seule ligne.

Le résultat est un tableau (nombre, vecteur ou matrice) de la même taille que le tableau input_variable.

Par exemple, le code suivant permet de définir la fonction :

$f(x) = e^{x^2}(lnx)^2$

```
>> f=@(x) exp(x.^2).*(log(x)).^2
      f =
            @(x)exp(x^2)*(log(x))^2
```

Pour calculer la valeur de f en x=2 ou pour v=[0 .1 :1.2 :7.7] on écrit ;

```
>>f(2)
      ans =
            26.2318
>>v=[0.1:1.2:7.7]
      v =
            0.1000
            1.3000
            2.5000
            3.7000
            4.9000
            6.1000
            7.3000
>>f(v)
      ans =
            0.0000
            0.0000
            0.0000
            0.0000
            0.0000
            0.0000
            5.4996
>>format short  %on change le format pour afficher plus de chiffres
>> format short
>> f(v)
      ans =
            5.3552e+00
            3.7305e-01
4.3492e+02
            1.5098e+06
            6.7575e+10
            4.7275e+16
            5.4996e+23
```

3.5 FICHIER .m

Un fichier L'essentiel du travail sous MATLAB consiste à créer ou modifier des fichiers avec l'extension .m qui est l'extension standard pour les procédures MATLAB. Lorsque l'on réalise une tache sous MATLAB, il est très souvent possible de le faire en utilisant uniquement la *Command Window*. Cependant lorsque cette tache devientplus complexe (plusieurs dizaines de ligne de code)ou que l'on souhaite pouvoir la transmettre à un autre utilisateur, on utilise la fenêtre *Editor* pour créer un fichier .m qui contiendra toutes les instructions à exécuter.

3.5.1 Edition de fichier :

Un fichier .m est soit un script ou une fonction (function en anglai). Un script est une suite de commande qui auraient pu être saisies directement dans la *Command Window*. Une fonction quant à elle permet d'étendre les possibilités au-delà des fonctions prédéfinies. MATLAB offre la possibilité. MATLAB offre la possibilité de sauvegarder une suite d'instructions dans des fichiers ayant l'extension .m, appelés fichiers .m. Il y a deux types des fichiers.m : les fichiers scripts et les fonctions.

Les instructions des fichiers function ne sont pas exécutées dans le Workspace courant mais dans un Workspace différent qui est crée à chaque fois que la fonction est appelée. Comme conséquence, on est obligés de transférer les variable de l'espace de travail Current Workspace vers Workspace de la fonction et de spécifier explicitement les variables de retour, en autres termes on doit spécifier les arguments d'entrée et de sortie.

Pour éditer un fichier .m, on choisit *New* ou *Open* à partir de File du menu. L'éditeur de MATLAB s'ouvre dans une fenêtre appelée *MATLAB Editor* dans laquelle on peut commencer à éditer un nouveau fichier ou charger en mémoire un fichier pré-existant sur le disque dur ou sur un autre support. Si on utilise l'option du menu New→ *function*,l'éditeur s'ouvre directement .Pour sauvegarder un ancien fichier il faut utiliser Save As. Le répertoire dans lequel les fichiers sont sauvegardés est appelé *Working Folder*

3.5.2 Fichiers fonctions .m :

La structure d'une fonction comporte nécessairement :

Le réservé fonction ;

- Le nom de la fonction, par exemple $mafonction$;
- Un ensemble de paramètres d'entrée $e1, e2, \ldots \ldots en;$
- Un ensemble de paramètres de sortie $s1, s2, \ldots \ldots sn;$

Les arguments d'entrée et sortie, s'ils existent, peuvent être un nombre, un vecteur, une matrice, une chaîne de caractères.

Exemple de lignes à saisir dans l'éditeur de MATLAB pour définir la fonction $mafonction$:

```
function [s1,s2, ... ...] = mafonction (e1,e2, ... ...)

% Mettre ici les lignes commentaires à afficher dans la fenêtre de commandes suite à
une demande d'aide avec help mafonction

instruction1

instruction2

.....

end
```

Exemple :

La fonction suivante appelée mafc permet le calcul de l'expression mathématique :

$f(x) = x^4 + 2x - 5/x$

```
Function   y=mafc(x)

% C'est ma première fonction

% La fonction n'est pas définie pour x=0

Y=x.^4+2*x-5./x ;

% deuxième série de commande
```

L'argument de sortie est y et l'argument d'entrée est x.

La fonction sous MATLAB à le nom 'mafc'. Il est préférable de sauvegarder cette fonction dans un fichier qui porte le même nom. D'ailleurs c'est le nom (En cliquant sur l'icône de sauvegarde représentée par une disquette ou bien en tapant

simultanément Ctrl+s).

La première série de commentaire sert à documenter la fonction, on y a accès avec la commande help mafc doc ('mafc').

Exemple

```
>>help mafc
```

C'est ma première fonction

La fonction n'est pas définie pour x=0

```
>>mafc(4)
```

```
ans=
        252.7500
```

Ou bien

```
>>x=4
```

```
x=
        4
```

```
>>mafc(x)
```

```
ans=
        252.7500
```

Le paramètre d'entrée x est dit muet ; on peut utiliser n'importe quel nom de variable lors de l'appel à la fonction.

Exemple :

```
>>z=4
```

```
z=
        4
```

```
>>mafc(z)
```

```
ans=
        252.7500
```

Le paramètre d'entrée x est une variable locale, on ne peut pas y accéder en dehors du fichier de la fonction. Dans l'exemple qui suit on autorise l'affichage de la valeur de x dans le corps de la fonction pour mettre en évidence la différence entre la variable x du *Current Workspace* et la variable x du *Workspace* de la fonction.

```
Function  y=mafc(x)

% On affiche la valeur de x à l'intérieure de la fonction x
```

```
y=x.^4+2*x-5./x ;
```

```
>>clear
>> x=10

      x=
            10
>>mafc(5)
      x=
            621
>> z=4
      z=
            4
>>x
      x=
            10
```

La valeur 10 attribuée dans la *Command Window* , à la variable x ne sera pas visible à l'intérieur du corps de la fonction . C'est la valeur 5 qui a été transférée à la fonction qui sera attribuée à x l'intérieur de la fonction. Lorsque la fonction est exécutée, on constate que la valeur de x définie dans la *Command Window* avant l'appel de la fonction n'a pas été modifiée.

Prenons un autre exemple pour illustrer la notion de variable locale. Considérons la fonction :

```
Function   y=mafc(x)

% On introduit un paramètre 'a' sans donner sa valeur

y=a*x.^4+2*x-5./x ;
```

Si on veut calculer la valeur de la fonction pour x=4, on obtient un message d'erreur.

```
>> x=10

      x=
            10
>>clear
>>mafc(4)
      undefined function or variable

      erreur in mafc (line 4)
            y=a*x.^4+2*x-5./x ;
```

Si dans la *Command Window* on donne à 'a' une valeur

```
>>a=1
```

On obtient le même massage d'erreur car la variable a est locale, elle est définie uniquement dans le *Workspace*.

Pour que a soit visible à l'intérieur de la fonction il faut déclarer a comme variable globale dans le *Workspace* et dans le corps de la fonction. Pour cela on utilise la commande *globale* dans la fonction *:*

```
Function   y=mafc(x)

global a

% On déclare 'a' comme variable globale

y=a*x.^4+2*x-5./x ;
```

Et également dans la *Command Window* :

```
>>a=1
```

Dans ce cas le calcul s'effectue.

```
>>globale a
>> a=1
```

```
>>mafc(4)
```

```
ans=
        252.7500
```

Exemple :

Ecrie un fichier function définissant la fonction mathématique

$$y(x) = \left[\frac{e^{-x}}{x^2+1} + sin^2(x)\right]^2 + 0.2$$

```
Function   y=mafc(x)

u=exp(-x)./(x.^2+1) ;

v=sin(x).^2 ;

w=u+v ;

y=w.^2+0.2 ;
```

Après avoir sauvegardé ce fichier sous le fun.m om peut évaluer la valeur de

fun pour une variable quelconque *var (nombre, vecteur ou matrice)* en écrivant *fun(var)* dans la fenêtre de commande. Notons que les variables u, v et w, définie à l'intérieur du corps de la fonction, ne seront pas renvoyées dans le *Command Workspace.*

Une fonction ne peut etre appelée que si elle est sauvegardée dans le répertoire de travail*(Working Folder)* ou dans l'un des répertoires déclarés dans *path.*

Dans le cas contraire l'appel de la fonction génère le message d'erreur :

```
Undefind function or variable 'mafc'
```

Pour accéder à une fonction sauvegardée dans un répertoire intitulé par exemple *monrepertoire* ,on signale ce répertoire et son chemin d'accès à l'aide de la commande :

```
>>addpath C:\chemin\monrepertoire
```

Dans une fonction on peut avoir plusieurs paramètres d'entrée et plusieurs paramètres de sortie.

Exemple :

On veut écrire une fonction permettant de passer des coordonnées cartésiennes (x,y) aux coordonnées polaires (ρ, θ).

Paramètres d'entrée

- Abscisse x

- Ordonnée y

Paramètres de sortie

- Rayon vecteur $\rho = \sqrt{x^2 + y^2}$

- Argument $\theta = \arctan\left(\frac{y}{x}\right)$

Pour réaliser cette opération, on écrit un fichier *polaire.m*contenant les lignes de commandes suivantes :

```
Function [r, theta]=polaire (x,y)
```

```
% fonction pour passer en coordonnées polaires
r=sqrt(x.^2+y.^2) ;
theta=atan(y./x) ;
```

Dans ce fichier :

- polaire : est le nom de la fonction,
- function : est le mot réservé pour la définition de la fonction,
- % : précède les commentaires accessibles à l'aide de help polaire,
- r, tetha : sont les paramètres de sortie
- x, y : sont les paramètres d'entrée

Pour l'utiliser à partir de la fenêtre MATLAB on écrit les instructions suivantes dans la fenêtre de commande :

- Si seul le rayon intéresse

```
>>r=polaire(2,3)
```

Ou bien

```
>>polaire(2,3)
ans=
        3.6055513
```

- Si on veut récupérer à la fois le rayon et l'angle

```
>>[r, t]=polaire(2,3)
r=
        0.932
t=
        3.6056
```

Exemple :

Calcul de la moyenne d'un ensemble de nombres. On écrit le fichier fonction suivant :

```
Function  y=moyenne (x)
% moyenne ou valeur moyenne
Y=sum(x)/length(x) ;
```

Exemple d'appel de moyenne :

```
>>z=1 :99
>>moyenne(z)
```

Exemple :

Un fonction peut avoir plusieurs paramètres d'entrée et plusieurs paramètres de sortie.

```
Function [A,C]=mafun2(E,F,G)

% A et C deux matrices de sortie

% E, F et G trois matrices d'entrée

% La matrice A et le produit des matrices E et F

A=E*F ;

% C correspond aux valeurs propres de G

C=eig(G) ;
```

3.6 SOUS-FONCTIONS

Dans un même fichier function on peut avoir plusieurs fonctions. La première fonction est appelée fonction principale et les autres fonctions locales ou sous-fonctions. Celles-ci ont la même structure que les fonctions définies précédemment, mais ne sont pas accessibles en dehors de la fonction principale. Les sous-fonctions favorisent le partage de lignes de commandes, sans multiplier inutilement les fichiers. L'ordre des fonctions n'a aucune importance. Les sous-fonctions et la fonction principale peuvent être définies dans n'importe quel ordre, à condition qu'elles soient dans le même fichier.

3.6.1 Fonctions locales

Une fonction locale est une fonction définie dans le même fichier que la fonction principale (portant le nom du fichier) mais qui n'est utilisable que localement dans la fonction principale, ou par d'autres fonctions locales de ce même fichier.

```
function y=mafct4(x)
%**********************Fonction principale : mafct4**********************
a=1.3 ;
b=2 ;
y=fct5(x)*(b*x)^2 ;
end
%**********************Fonction secondaire : fct5**********************
function z=fct5=(t)
Z=log(t)/t ;
end
```

La première fonction mafct4 fait appel à la deuxième fct5. Les noms des paramètres ne sont pas forcément les mêmes, puisqu'ils sont muets. La commande end indique le début et la fin de chaque fonction.

On peut faire à plusieurs fonctions écritures dans le même fichier.

Exemple :

Le fichier function ci-dessous appelé statistiques .m contient une fonction principale, statistiques, et deux fonctions locales moyenne et médiane.

```
%**********************Fonction principale : statistiques **********************
Function [avg,med]= statistiques(x)
n=length(x) ;
avg=moyenne(x,n) ;
med=mediane(x,n) ;
end
%**********************Fonction locale moyenne **********************
```

```
function a=moyenne(v,n)

a=sum(v)/n ;

end

%**********************Fonction locale dediane **********************

function m=mediane(v,n)

w=sort(v);

if rem(n,2) ==1

      m=w((n+1)/2) ;

else

     m=(w(n/2)+w(n/2+1)) /2;

end

end
```

Exemple :

```
>>x=rand(1,100) ;
>>[moy med]=statistiques(x)
       moy=
              0.4723
       med=
              0.4487
```

3.6.2 Fonctions imbriquées

Une fonction imbriquée (ou nested function) est une fonction qui est complément contenue à l'intérieur d'un fonction appelé fonction parent. Par exemple, la fonction ci-dessous appelée fparent contient une fonction imbriquée appelée fimbrique

```
function fparent

disp('Nous sommes dans la fonction parent')

fimbrique
```

```
function fimbrique
disp('Nous sommes dans la fonctionimbriquée)
end
end
```

La différence essentielle entre les fonctions imbriquées et les autres types de fonction et qu'une fonction imbriquée peut accéder à des variables définies dans la fonction qui l'englobe, et les modifier. Les fonctions imbriquée peuvent ainsi modifier des variables qui n'ont pas été transmises en tant que paramètres lors de leur appel.

3.7 FONCTIONS DE FONCTIONS

Le paramètre d'entrée d'une fonction peut lui-même être une fonction.

Exemple :

```
>>f=@(x)exp./(1+x.^2)
>>integral(f,-1,1)
>>integral(f,Inf,0)
>>integral(f,0,Inf)
```

Commentaires :

- La ligne de commande integral(f,a,b) calcule l'intégrale de f entre a et b.
- Nous utilisons ici les opérations. /et. car est nécessaire de pouvoir appliquer, terme à terme, la fonction f à une matrice.

Exemple :

Utilisation de la fonction fzero pour trouver une racine d'une fonction d'une variable.

```
x=fzero(fun,x0)
x=fzero(fun,x0,options)
x=fzero(fun,x0,options, P1, P2, . . .)
[x,fval]=fzero(...)
[x,fval,exitflag]=fzero(...)
[x,fval,exitflag, output]=fzero(...)
```

Descriptions :

- Fun peut aussi bien être une fonction inline, une fonction prédéfinie MATLab, une fonction anonyme ou une fonction définie par un fichier function.
- x=fzero(fun,x0) trouve un zéro près de x0, si x0 est un scalaire. La valeur x renvoyée par fzero est prés d'un point où fun change de signe ou bien NaN si la recherche a échoué.
- Si x0 est un vecteur à 2composantes , fzero le considère come un intervalle tel que fun (x0 (1)) et fun (x0(2)) sont de singes opposées. Si ce n'est pas le cas il y a une erreur.
- x = fzero (fun,x0,[],P1,P2…) permet de passer des paramètres supplémentaires P1 ,P2, etc ,…à la fonction fun.

Exemple :

1- Trouver les zéros de la fonction p(x)=x^2-x-2

```
>> P=inline('x^2-x-2') ;
      P=
      Inline function
p(x)=x^2-x-2

>> x=fzero (p,-2.5) ;

      x=
            -1
>> x=fzero (p,3) ;

      x=
            2
```

Les zéros de p(x)=x^2-x-2sont x=-1 et x=2,

2- Trouve un zéro de sin (x) au voisinage de x=3

```
fun=@sin ; % définition da la fonction
x0=3 ; % point initial
x=fzero(fun,x0)
x=
3.1416
```

3- Trouve le zéro de la fonction cosinus entre 1 et 2.

```
fun=@cos ; % définition da la fonction
x0=[1 2] ; % intervalle initial
x=fzero(fun,x0)
x=
    1.5708
```

Notons que cos (1) et cos (2) sont de singes différents.

4- Racine d'une fonction définie par un fichier function . Par exemple, pour trouver les zéros de F(x) =x^3-2x-5

- On crée d'abord dans l'éditeur un fichier appelé f.m

```
function y=f(x)
y=x.^3-2*x-5 ;
```

- On sauvegarde ce fichier dans le répertoire de travail sous le nom f.m
- Pour recherche le zéro au voisinage de x=2 , on écrit dans la fenêtre de commande :

```
fun=@f ; % définition da la fonction
x0=2; % initial initial
x=fzero(fun,x0)
x=
2.0946
```

Si l'on cherche plusieurs racines, il sera nécessaire de modifier la valeur d'initialisation pour espérer trouver de nouvelles racines, d'où l'importance du choix de la valeur d'initialisation de la recherche. Une description plus détaillée de la fonction fzeo est disponible dans l'aide en ligne.

3.8 EXERCICES

EXERCICE 01 :

Définir la fonction $h(x) = xsin\left(\frac{\pi x}{2}\right)$

- Sous la forme d'une fonction inline
- Sous la forme d'une fonction anonyme

EXERCICE 02 :

Comment définir l'expression mathématique$g(x) = \int_0^{2\pi} xsin\left(\frac{\pi x}{2}\right)$?

EXERCICE 03 :

Utiliser la commande quad1 puis la commande integral pour calculer les intégrales suivantes :

1. $\int_0^{\pi} e^{sinx}\, dx$
2. $\int_0^1 \sqrt{x^3+1}dx$

EXERCICE 04 :

Proposer deux méthodes différentes d'utilisation de la commande fzero pour trouver les zeros de $g(x) = x - \cos(x)$ au voisinage de x=1.

EXERCICE 05 :

1. Donnez les commandes Matlab nécessaires pour dessiner les courbes des deux fonctions suivantes (séparément):
   1) f(x) $= -2x^2 + 3x - 1$ pour x ∈ [-4 , 4], pas = 0.2 ;
   2) g(x) $= x\, sin(x)$ pour x ∈ $\left[\frac{-5\pi}{2}, \frac{5\pi}{2}\right]$, pas = $\frac{\pi}{12}$;
2. Donnez les commandes pour dessinez la courbe de f(x) en changeant les limites des axes des abscisses en [-1,3] et les limites des axes des ordonnées en [-20,5]

EXERCICE 06 :

Donnez les commandes Matlab nécessaires pour dessiner la courbe de la fonction:

f(x) $= \frac{-2e^x - 2x^2}{3-5x^3}$ pour x ∈ [-1 , 3], pas = 0.1 ;

>>.....

>>.....

>>.....

EXERCICE 07 :

1. Donnez les commandes Matlab nécessaires pour dessiner les courbes des deux fonctions suivantes :
   1) $f(x) = 3x^5 - 6x^2 + 2x - 1$ pour x ∈ [-7 , 7], pas = 0.2
   2) $g(x) = x \sin(x)$ pour x ∈ $[-2\pi, 2\pi]$, pas = $\frac{\pi}{9}$

>> .....

>> .....

>> .....

>> .....

>> .....

>> .....

EXERCICE 08 :

Voici un programme Matlab qui calcule la somme suivante :

$$\sum_{k=0}^{n} \frac{(-1^k)k}{k+1} = 0 - \frac{1}{2} + \frac{2}{3} - \frac{3}{4} + \cdots + \frac{(-1^n)n}{n+1}$$

```
n = input('Entrez un nombre naturel: ');
Somme = 0;
for k = 0:n
    Somme = Somme + (-1)^k * k/(k+1);
end
disp(Somme)
```

1) Remplacez l'instruction **for** par **while** en préservant la fonctionnalité.

.....

.....

.....

.....

2) Transformez ce programme en une fonction nommée **sommeFct**.

.....

.....

.....

.....

3) Ecrire un programme qui calcule la somme suivante :

$$\sum_{k=0}^{n} \frac{n}{2^k} = n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^n}$$

.....

.....

CHAPITRE 4

LES POLYNOMES

4 Chapitre 04 : Les polynômes

4.1 DEFINITION :

Un polynôme de la forme $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots a_2 x^2 + a_1 x + a_0$ est représenté en MATLAB par un vecteur ligne $[a_n, a_{n-1} \ldots a_2, a_1, a_0]$ (ou vecteur colonne).

Afin d'évaluer un polynôme pour une valeur fixe a, on utilise la commande polyval(P,a).

```
>>P=[1 2 1]      %définir le polynôme x^2+2x+1
```

```
P=
     1   2   1
```

```
>>polyval(P,0),polyval(P,1),polyval(P,2)
```

```
ans=
     1     % P(0)
ans=
     4     % P(1)
ans=
     9     % P(2)
```

4.2 OPERATIONS ARITHMETIQUES

- Matlab fournie deux opérations arithmétiques de multiplication et de division à travers les commande conv et deconv. Il reste au programmeur de programmer les deux autres opérations d'addition et de soustraction.
- C = conv(A, B) est la convolution des tableaux A et B, c'est à dire les coefficients du produit des deux polynômes.
- [Q, R] = deconv(A, B) est la dé-convolution des tableaux A et B où Q est lequotient de la division et R est le reste (A = conv(B ,Q) + R).

```
>>P1=[1 2 1],P2=[1 1],      %définir les polynômesP1=x^2+2x+1, P2=x+1
```

```
P1=
12   1
P2=
1   1
```

```
>>conv(P1,P2)   % P1*P2
```

```
ans=
        1     2     2     1    %P1*P2=x^3+2x^2+2x+1
>> P1=[Q R]=deconv(P1,P2)          %P1/P2

Q=
        1     0              % Quotient=x
R=
        0     0     1        % Reste=1
```

4.3 RACINES ET INTERPOLATIONS

- La commande roots(P) fait extraire les racines d'un polynôme P.
- La commande poly(V) retourne un polynôme depuis ses racines stockées dans le vecteur V.

```
>>P=[1 -5 6]           %définir le polynômeP

P1=
1  -5  6
>>Roots(P)

ans=
        3.0000
        2.0000
>>poly([2 3])

P1=
     1  -5  6
```

La commande Polyfit(X, Y, n) permet d'approximer un polynôme de dégrée n qui passent approximativement par les points (X, Y). Cette approximation P(X(i))=Y(i) est au sens des moindres carrées : trouver P tel que $\sqrt{\sum_i (P(x_i)+y_i)^2}$ est minimal

```
>>X=[0 1 2],Y=[-1 0 3],          %définir les polynômes X,Y

X=
        0  1  2
Y=
       -1  0  3
>>polyfit(X,Y,0)

ans=
        0.6667              %P(x)=2/3 (avec erreur)


>>polyfit(X,Y,1)

ans=
```

```
            2.0000      -1.3333                  %P(x)=2x-1/3 (avec erreur)
>>polyfit(X,Y,2)
```

ans=

```
            1.0000      0       -1.0000          %P(x)=x^2-1 (meilleur)
```

4.4 DERIVATION ET INTEGRATION D'UN POLYNOME

- La commande polyder(P) retourne la dérivée d'un polynôme P.
- La commande polyint(P) retourne l'intégral d'un polynôme P. Si on veut calculer l'intégral entre deux point x1 et x2, on utilise la commande polyvalpour évaluer l'intégral sur ces deux points puis on applique la soustraction.

```
>>P=[1 2 1]          %définir le polynôme P(x)=x^2+2x+1
```

P=

```
1   2   1
>>Pder=polyder(P)          %définir le polynôme P'(x)=2x+2
```

Pder=

```
2   2
>>Pint=polyint(P)          %integral (P(x))=1/3x^3+x^2+x
```

Pint=

```
            0.3333      1.0000      1.0000      0
>>polyval(Pint,2)-polyval(Pint,1)%integral (P(x)) entre 1 et 2
```

ans=

```
6.3333
```

CHAPITRE 5

INTRODUCTION A LA PROGRAMMATION AVEC MATLAB

5 Chapitre 05 : Introduction à la programmation avec Matlab

L'utilisation Matlab effectue des commandes pour évaluer des expressions en les écrivant dans la ligne de commande (Après le prompt >>), par conséquent les commandes présenté s'écrivent généralement sous forme d'une seule instruction (éventuellement sur une seule ligne).

Il existe des problèmes dont la description de leurs solutions nécessite plusieurs instructions, ce qui réclame l'utilisation de plusieurs lignes. Comme par exemple la recherche des racines d'une équation de second degré (avec prise en compte de tous les cas possibles).

Une collection d'instructions bien structurées visant à résoudre un problème donnée s'appelle un programme. Dans cette partie du cours, on va présenter les mécanismes d'écriture et d'exécution des programmes en Matlab.

5.1 Généralités :

5.1.1 Les commentaires

Les commentaires sont des phrases explicatives ignorées par Matlab et destinées pour l'utilisateur afin de l'aider à comprendre la partie du code commentée.

En Matlab un commentaire commence par le symbole % et occupe le reste de la ligne.

Par exemple :

```
>> A=B+C ;              %Donner à A la valeur de B+C
```

5.1.2 Écriture des expressions longues :

Si l'écriture d'une expression longue ne peut pas être enclavée dans une seule ligne, il est possible de la diviser en plusieurs lignes en mettant à la fin de chaque

ligne au moins trois points.

Exemple :

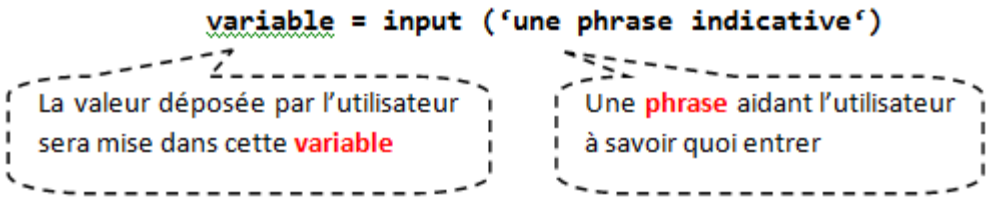
```
>> (sin(pi/3)^2/cos(pi/3)^2)-(1-2*(5+sqrt(x)^5/(-2*x^3-x^2)^1+3*x)) ;
```

Cette expression peut être réécrite de la façon suivante :

```
>> (sin(pi/3)^2/cos(pi/3)^2)- ...  ↵
>> (1-2*(5+sqrt(x)^5 .....          ↵
>>/(-2*x^3-x^2)^1+3*x)) ;          ↵
```

5.1.3 Lecture des données dans un programme (Les entrées) :

Pour lire une valeur donnée par l'utilisateur, il est possible d'utiliser la commande ***input***, qui a la syntaxe suivante :



Quand Matlab exécute une telle instruction, La phrase indicative sera affichée à l'utilisateur en attendant que ce dernier entre une valeur.

Par exemple :

```
>> A = input ('Entrez un nombre entier : ')          ↵
   Entrez un nombre entier : 5                       ↵
   A =
      5
>>

>> A = input ('Entrez un nombre entier : ');         ↵
   Entrez un nombre entier : 5                       ↵
>>

>> B = input ('Entrez un vecteur ligne : ')          ↵
   Entrez un vecteur ligne : [1:2:8,3:-1:0]          ↵
   B =
      1     3     5     7     3     2     1     0
```

5.1.4 Ecriture des données dans un programme (Les sorties) :

On a déjà vu que Matlab peut afficher la valeur d'une variable en tapant seulement le nom de cette dernière. Par exemple :

```
>> A = 5 ;
```

```
>> A                    %Demander à Matlab d'afficher la valeur de A
   A =
        5
```

Avec cette méthode, Matlab écrit le nom de la variable (A) puis le signe (=) suivie de la valeur désirée. Cependant, il existe des cas où on désire afficher uniquement la valeur de la variable (sans le nom et sans le signe =).

Pour cela, on peut utiliser la fonction ***disp***, et qui a la syntaxe suivante : ***disp (objet)***

La valeur de l'objet peut être un nombre, un vecteur, une matrice, une chaine de caractères ou une expression.

On signale qu'avec un vecteur ou une matrice vide, ***disp*** n'affiche rien.

Exemple :

```
>>disp(A)                %Afficher la valeur de A sans 'A = '
      5
>>disp(A);                       %Le point virgule n'a pas d'effet
      5
>> B                             % Afficher le vecteur B par la méthode classique
   B =
      1     3     5     7     3     2     1     0
>>disp(B)                % Afficher le vecteur B sans 'B = '
      1     3     5     7     3     2     1     0
>> C = 3 :1 :0                   %Création d'un vecteur C vide
   C =
   Empty matrix: 1-by-0
>>disp(C)                % disp n'affiche rien si le vecteur est vide
>>
```

5.2 Les expressions logiques :

5.2.1 Les opérations logiques :

L'opération de comparaison	***Sa signification***
==	l'égalité
~=	l'inégalité
>	supérieur à
<	inferieur à
>=	supérieur ou égale à
<=	inferieur ou égale à
L'opération logique	***Sa signification***
&	le **et** logique

|	le **ou** logique
~	la **négation** logique

En Matlab une variable logique peut prendre les valeurs **1**(vrai) ou **0**(faux) avec une petite règle qui admette que :

- Toute valeur égale à 0 sera considérée comme fausse **(= 0 ⇒ Faux)**
- Toute valeur différente de 0 sera considérée comme vrai **(≠ 0 ⇒ Vrai)**.

Le tableau suivant résume le fonctionnement des opérations logiques :

a	*b*	*a & b*	*a | b*	*~a*
1 *(vrai)*	1 *(vrai)*	1	1	0
1 *(vrai)*	0 *(faux)*	0	1	0
0 *(faux)*	1 *(vrai)*	0	1	1
0 *(faux)*	0 *(faux)*	0	0	1

Par exemple :

```
>> x=10;
>> y=20;
>> x < y              % affiche 1 (vrai)
       ans =
             1
>> x <= 10            %affiche 1 (vrai)
       ans =
             1
>>x == y              %affiche 0 (faux)
       ans =
             0
>> (0 < x) & (y < 30)            % affiche 1 (vrai)
       ans =
             1
>> (x > 10) | (y > 100)          %affiche 0 (faux)
       ans =
             0
>> ~(x > 10)          %affiche 1 (vrai)
       ans =
             1
>> 10 & 1                        %10 est considéré comme vrai donc 1 & 1 = 1
       ans =
             1
>> 10 & 0                        %1 & 0 = 1
       ans =
             0
```

5.2.2 Comparaison des matrices :

La comparaison des vecteurs et des matrices diffère quelque peu des scalaires, d'où l'utilité des deux fonctions '*isequal*' et '*isempty*' (qui permettent de donner une réponse concise pour la comparaison).

La fonction	*Description*
isequal	teste si deux (ou plusieurs) matrices sont égales (ayant les mêmes éléments partout). Elle renvoie 1 si c'est le cas, et 0 sinon.
isempty	teste si une matrice est vide (ne contient aucun élément). Elle renvoie 1 si c'est le cas, et 0 sinon.

Pour mieux percevoir l'impact de ces fonctions suivons l'exemple suivant :

```
>> A = [5,2;-1,3]          %Créer la matrice A
      A =
             5     2
            -1     3
>> B = [5,1;0,3]           %Créer la matrice B
      B =
            5     1
            0     3
>> A==B                    % Tester si A=B ? (1 ou 0 selon la position)
      ans =
            1     0
            0     1
>>isequal(A,B)             % Tester si effectivement A et B sont égales (les mêmes)
      ans =
            0
>> C=[] ;                  %Créer la matrice vide C
>>isempty(C)               %Tester si C est vide (affiche vrai = 1)
      ans =
            1
>>isempty(A)                %Tester si A est vide (affiche faux = 0)
      ans =
            0
```

5.3 Structures de contrôle de flux :

Les structures de contrôle de flux sont des instructions permettant de définir et de manipuler l'ordre d'exécution des taches dans un programme. Elles offrent la possibilité de réaliser des traitements différents selon l'état des données du programme, ou de réaliser des boucles répétitives pour un processus donnée.

Matlab compte huit structures de control de flux à savoir :

- **if**
- **switch**
- **for**
- **while**
- continue
- break
- try - catch

- return

Nous exposons les quatre premières : (if, switch, for et while)

5.3.1 L'instruction if :

L'instruction **if** est la plus simple et la plus utilisée des structures de contrôle de flux. Elle permette d'orienter l'exécution du programme en fonction de la valeur logique d'une condition. Sa syntaxe générale est la suivante :

```
if (condition)
       instruction_1
       instruction_2
                . . .
       Instruction_N
end
```

ou bien

```
if (condition)
       ensemble d'instructions 1
else
       ensemble d'instructions 2
end
```

Si la condition est évaluée à vari, les instructions entre le **if** et le **end** seront exécutées, sinon elles ne seront pas (ou si un **else** existe les instructions entre le **else** et le **end** seront exécutées). S'il est nécessaire de vérifier plusieurs conditions au lieu d'une seule, on peut utiliser des clauses **elseif** pour chaque nouvelle condition, et à la fin on peut mettre un **else** dans le cas ou aucune condition n'a été évaluée à vrai. Voici donc la syntaxe générale :

```
if (expression_1)
       Ensemble d'instructions 1
elseif (expression_2)
       Ensemble d'instructions 2
             ....
elseif (expression_n)
       Ensemble d'instructions n
else
       Ensemble d'instructions si toutes les expressions étaient fausses
end
```

Par exemple, le programme suivant vous définit selon votre âge :

```
>> age = input('Entrez votre âge : '); ...
if (age < 2)
disp('Vous êtes un bébé')
elseif (age < 13)
disp('Vous êtes un enfant')
elseif (age < 18)
disp ('Vous êtes un adolescent')
```

```
elseif (age < 60)
disp ('Vous êtes un adulte)
else
disp ('Vous êtes un vieillard')
end
```

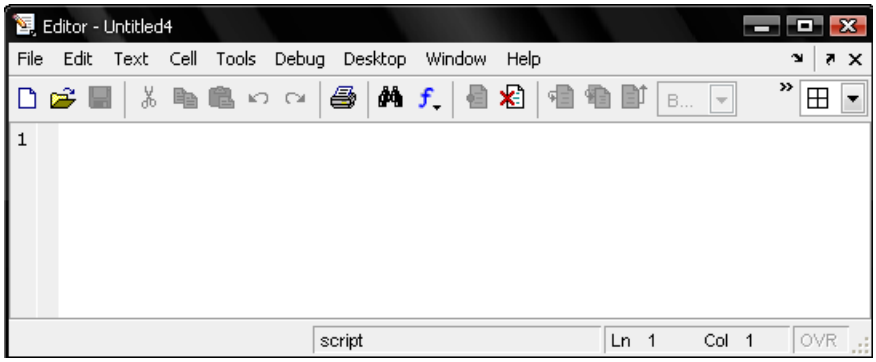
Comme vous pouvez le constatez, l'écriture d'un programme Matlab directement après l'invité de commande (le prompt >>) est un peu déplaisant et ennuyeux.

Une méthode plus pratique consiste à écrire le programme dans un fichier séparé, et d'appeler ce programme (au besoin) en tapant le nom du fichier dans l'invité de commande.

Cette approche est définie en Matlab par les M-Files, qui sont des fichiers pouvant contenir les données, les programmes (scripts) ou les fonctions que nous développons.

Pour créer un M-Files il suffit de taper la commande ***edit***, ou tout simplement aller dans le menu : File → New → M-Files (ou cliquer sur l'icône).

Dans tous les cas une fenêtre d'édition comme celui ci va apparaitre :



Tout ce qui vous reste à faire et d'écrire votre programme dans cette fenêtre, puis l'enregistrer avec un nom (par exemple : 'Premier_Programme.m'). On signale que l'extension des fichiers M-Files est toujours '.m'.

Maintenant, si nous voulons exécuter notre programme, il suffit d'aller à l'invité de commande habituel (>>) puis taper le nom de notre fichier (sans le '.m') comme ceci :

```
>> Premier_Programme ↵
```

Et l'exécution du programme va démarrer immédiatement.

Pour retourner à la fenêtre d'édition (après l'avoir fermer) il suffit de saisir la commande :

```
>>edit Premier_Programme        ↵
```

Exemple :

Créons un programme qui trouve les racines d'une équation de second degré désigné par :

$ax^2+bx+c=0$. Voici le M-File qui contient le programme (il est enregistré avec le nom 'Equation2deg.m')

```
% Programme de résolution de l'équation a*x^2+b*x+c=0

a = input ('Entrez la valeur de a : ');            % lire a
b = input ('Entrez la valeur de b : ');            %lire b
c = input ('Entrez la valeur de c : ');            %lire c

delta = b^2-4*a*c ;                              %Calculer delta
if delta<0
disp('Pas de solution')                          %Pas de solution
elseif delta==0
disp('Solution double : ')                       %Solution double
   x=-b/(2*a)
else
disp('Deux solutions distinctes: ')              %Deux solutions
x1=(-b+sqrt(delta))/(2*a)
x2=(-b-sqrt(delta))/(2*a)
end
```

Si nous voulons exécuter le programme, il suffit de taper le nom du programme :

```
>> Equation2deg                          ↵
      Entrez la valeur de a : -2         ↵
      Entrez la valeur de b : 1          ↵
      Entrez la valeur de c : 3          ↵
      Deux solutions :
      x1 =
            -1
      x2 =
            1.5000
```

Ainsi, le programme va être exécuté en suivant les instructions écrites dans son M-File. Si une instruction est terminée par un point virgule, alors la valeur de la variable concernée ne sera pas affichée, par contre si elle termine par une

virgule ou un saut à la ligne, alors les résultats seront affichés.

Remarque : Il existe la fonction ***solve*** prédéfinie en Matlab pour trouver les racines d'une équation (et beaucoup plus). Si nous voulons l'appliquer sur notre exemple, il suffit d'écrire :

```
>>solve('-2*x^2+x+3=0','x')
 ans =
 -1
        3/2
```

5.3.2 L'instruction switch :

L'instruction **switch** exécute des groupes d'instructions selon la valeur d'une variable ou d'une expression. Chaque groupe est associé a une clause **case** qui définie si ce groupe doit être exécuté ou pas selon l'égalité de la valeur de ce **case** avec le résultat d'évaluation de l'expression de **switch**. Si tous les **case** n'ont pas étés acceptés, il est possible d'ajouter une clause **otherwise** qui sera exécutée seulement si aucun **case** n'est exécuté.

Donc, la forme générale de cette instruction est :

```
switch (expression)
       case valeur_1
              Groupe d'instructions 1
       case valeur_2
              Groupe d'instructions 2
              . . .
       case valeur_n
              Groupe d'instructions n
       otherwise
              Groupe d'instructions si tous les case ont échoué
end
```

Exemple :

```
x = input ('Entrez un nombre : ') ;
switch(x)
       case 0
              disp('x = 0 ')
       case 10
              disp('x = 10 ')
       case 100
              disp('x = 100 ')
       otherwise
              disp('x n''est pas 0 ou 10 ou 100')
end
```

L'exécution va donner :

```
Entrez un nombre : 50          ↵
x n'est pas 0 ou 10 ou 100
```

5.3.3 L'instruction for :

L'instruction **for** répète l'exécution d'un groupe d'instructions un nombre déterminé de fois. Elle a la forme générale suivante :

```
for variable = expression_vecteur
      Groupe d'instructions
end
```

L'expression_vecteur corresponde à la définition d'un vecteur : *début : pas : fin* ou *début : fin*

Le variable va parcourir tous les éléments du vecteur défini par l'expression, et pour chacun il va exécuter le groupe d'instructions.

Exemple :

Dans le tableau suivant, nous avons trois formes de l'instruction **for** avec le résultat Matlab :

L'instruction for	`for i = 1 : 4`
	`  j=i*2 ;`
	`  disp(j)`
	`end`
		`for i = 1 : 2 : 4`
		`  j=i*2 ;`
		`  disp(j)`
		`end`
			`for i = [1,4,7]`
			`  j=i*2 ;`
			`  disp(j)`
			`end`

Le résultat de l'exécution	2 4 6 8	2 6	2 8 14

5.3.4 L'instruction While :

L'instruction **while** répète l'exécution d'un groupe d'instructions un nombre indéterminé de fois selon la valeur d'une condition logique. Elle a la forme générale suivante :

```
while (condition)
      Ensemble d'instructions
end
```

Tant que l'expression de **while** est évaluée à true, l'ensemble d'instructions s'exécutera en boucle.

Exemple :

```
a=1 ;
while (a~=0)
      a = input ('Entrez un nombre (0 pour terminer) : ') ;
end
```

Ce programme demande à l'utilisateur d'entrer un nombre. Si ce nombre n'est pas égal à 0 alors la boucle se répète, sinon (si la valeur donnée est 0) alors le programme s'arrête.

5.4 Exercice récapitulatif :

Il existe des fonctions prédéfinis en Matlab donnée dans le tableau ci-dessous. Essayons de les programmer (pour un vecteur donnée V).

La fonction	***Description***	***Le programme qui la simule***

sum (V)	La somme des éléments d'un vecteur V	`n = length(V);`
		`somme = 0 ;`
		`for i = 1 : n`
		`somme=somme+V(i) ;`
		`end`
		`disp(somme)`

prod (V)	Le produit des éléments d'un vecteur V	`n = length(V);`
		`produit = 1 ;`
		`for i = 1 : n`
		`produit=produit*V(i) ;`
		`end`
		`disp(produit)`

mean (V)	La moyenne des éléments d'un vecteur V	`n = length(V);`
		`moyenne = 0 ;`
		`for i = 1 : n`
		`moyenne = moyenne+V(i) ;`
		`end`
		`moyenne = moyenne / n`

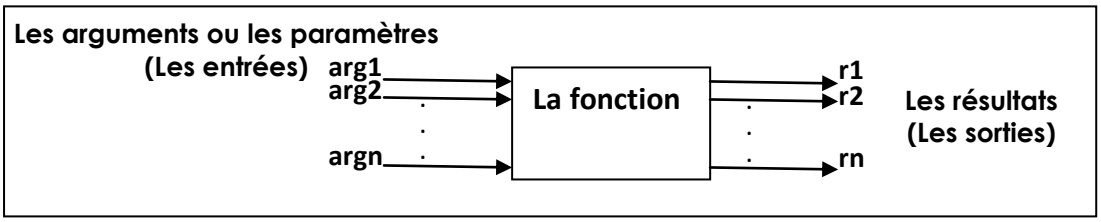
diag (V)	Créer une matrice ayant le vecteur V dans le diagonale, et 0 ailleurs	`n = length(V);`
		`A = zeros(n) ;`
		`for i = 1 : n`
		`  A(i,i)=V(i) ;`
		`end`
		`disp(A)`

sort (V)	Ordonne les éléments du vecteur V par ordre croissant	`n = length(V);`
		`for i = 1 : n-1`
		`  for j = i+1 : n`
		`  if V(i) > V(j)`
		`  tmp = V(i) ;`
		`  V(i) = V(j) ;`
		`  V(j) = tmp ;`
		`  end`
		`  end`
		`end`
		`disp(V)`

5.5 Les fonctions :

Il existe une différence de concept entre les fonctions en informatique ou en mathématique:

1. En informatique, une fonction est une routine (un sous programme) qui accepte des arguments (des paramètres) et qui renvoie un résultat.



2. En mathématique une fonction **f** est une relation qui attribue à chaque valeur x au plus une seule valeur **f(x)**.

5.5.1 Création d'une fonction dans un M-Files :

Matlab contient un grand nombre de fonctions prédéfinies comme ***sin***, ***cos***, ***sqrt***, ***sum***, ...etc. Et il est possible de créer nos propres fonctions en écrivant leurs codes source dans des fichiers M-Files (portant le même nom de fonction) en respectant la syntaxe suivante :

```
function [r1, r2, …, rn] = nom_fonction (arg1, arg2, …, argn)

      % le corps de la fonction
      . . .
      r1 = . . .   %la valeur retournée pour r1
      r2 = . . .   %la valeur retournée pour r2
      . . .
      rn = . . .   %la valeur retournée pour rn
end                %le end est facultatif
```

Ou : $\mathbf{r_1 ... r_n}$ sont les valeurs retournées, et $\mathbf{arg_1 ... arg_n}$ sont les arguments.

Exemple : Ecrire une fonction qui calcule la racine carrée d'un nombre par la méthode de Newton .

Solution :

```
>>edit
```

Le fichier racine.m

```
function r = racine(nombre)
    r = nombre/2;
precision = 6;
for i = 1:precision
        r = (r + nombre ./ r) / 2;
end
```

L'exécution :

```
>> x = racine(9)
      x =
           3
>> x = racine(196)
      x =
         14.0000
>> x = racine([16,144,9,5])
      x =
           4.0000   12.0000    3.0000    2.2361
```

Remarque :

Contrairement à un programme (un script), une fonction peut être utilisée dans une expression par exemple : **2*racine(9)-1.**

Comparaison entre un programme est une fonction

Un programme	*Une fonction*

`a = input('Entrez un nombre positif: ');`
`x = a/2;`
`precision = 6;`
`for i = 1:precision`
`  x = (x + a ./ x) / 2;`
`end`
`disp(x)`
	`function r = racine(nombre)`
	`  r = nombre/2;`
	`precision = 6;`
	`for i = 1:precision`
	`  r = (r + nombre ./ r) / 2;`
	`end`

L'exécution :
`>> racine` ↵
`Entrez un nombre positif: 16` ↵
`4`
	L'exécution :
	`>> racine(16)`
	`ans =`
	`4`

on ne peut pas écrire des expressions tel que : **>> 2 * racine + 4** ✗	on peut écrire sans problème des expressions comme : **>> 2 * racine(x) + 4** ✓

Exercice 01 :

Ce programme calcule la racine carrée d'un nombre par la méthode de Newton.

1) Exécuter manuellement le programme pour a=16, a=5, a=169.

2) Remplacer l'instruction **for** par l'instruction **while** en préservant la fonctionnalité du programme.

3) Modifier le programme pour qu'il soit applicable sur un vecteur et pas uniquement sur un nombre.

4) Quel est l'impact sur le résultat du programme si on choisit une valeur trop petite pour la variable **precision** (par exemple **precision = 3**) ?

```
a = input('Entrez un nombre positif: ');
x = a/2;
precision = 6;
for i = 1:precision
    x = (x + a / x) / 2;
end
disp(x)
```

Exercice 02 :

1) Ecrivez un programme qui calcule le factoriel d'un nombre entier n (n!).
2) Ecrivez un programme qui calcule les deux sommes suivantes :

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}, \qquad et \qquad \sum_{k=1}^{n} \frac{(-1)^k}{k^2} = -1 + \frac{1}{4} - \frac{1}{9} + \cdots \mp \frac{1}{n^2}, \qquad n > 0 \ entier$$

3) En utilisant la fonction **mod** qui calcule le modulo (le reste de la division entière), écrivez une fonction qui peut indiquer si un nombre entier **a** est premier ou pas.

CHAPITRE 6

LES GRAPHIQUES ET LA VISUALISATION DES DONNEES EN MATLAB

6 Chapitre 06 : Les graphiques et la visualisation des données en MATLAB

Partant du principe qu'une image vaut mieux qu'un long discours, Matlab offre un puissant système de visualisation qui permet la présentation et l'affichage graphique des données d'une manière à la fois efficace et facile.

Dans cette partie du cours, nous allons présenter les principes de base indispensables pour dessiner des courbes en Matlab.

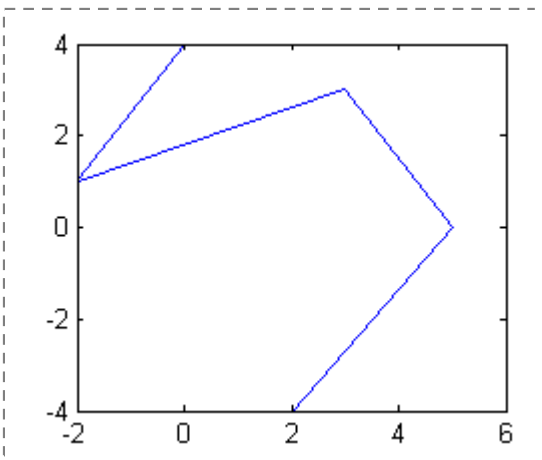
6.1 LA FONCTION PLOT :

La fonction ***plot*** est utilisable avec des vecteurs ou des matrices. Elle trace des lignes en reliant des points de coordonnées définies dans ses arguments, et elle à plusieurs formes :

- ***Si elle contient deux vecteurs de la même taille comme arguments*** : elle considère les valeurs du premier vecteur comme les éléments de l'axe X (les abscisses), et les valeurs du deuxième vecteur comme les éléments de l'axe Y (les ordonnées).

Exemple :

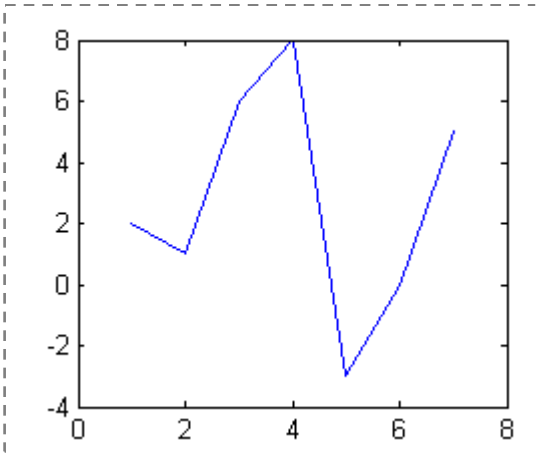
```
>> A = [2, 5, 3,-2,0]
A =
       2     5     3    -2     0

>> B = [-4, 0, 3, 1,4]
B =
      -4     0     3     1     4

>>plot(A,B)
```



❖ ***Si elle contient un seul vecteur comme argument :*** elle considère les valeurs du vecteur comme les éléments de l'axe Y (les ordonnées), et leurs positions relatives définirons l'axe X (les abscisses).

Exemple :

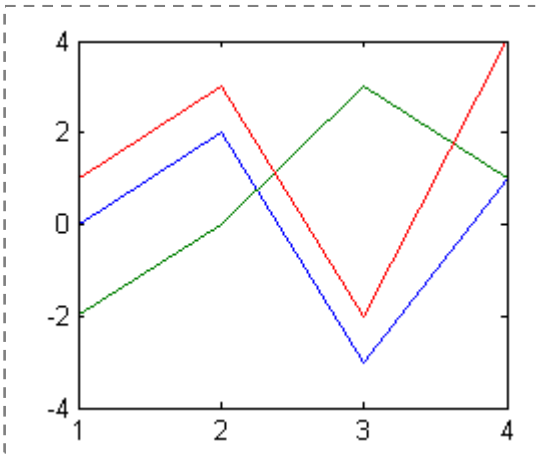
```
>> V = [2, 1, 6, 8,-3, 0,5]
V =
      2   1   6   8  -3   0   5

>>plot(V)
```



❖ ***Si elle contient une seule matrice comme argument :*** elle considère les valeurs de chaque colonne comme les éléments de l'axe Y, et leurs positions relatives (le numéro de ligne) comme les valeurs de l'axe X. Donc, elle donnera plusieurs courbes (une pour chaque colonne).

Exemple :

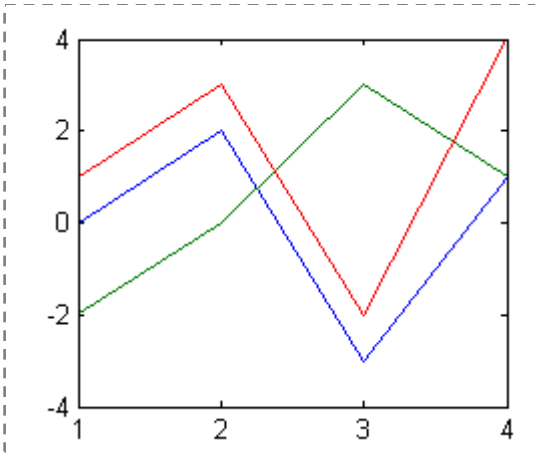
```
>> M = [0 -2 1;2 0 3;-3 3 -2;1 1 4]
M =
     0    -2     1
     2     0     3
    -3     3    -2
     1     1     4

>>plot(M)
```



❖ ***Si elle contient deux matrices comme arguments :*** elle considère les valeurs de chaque colonne de la première matrice comme les éléments de l'axe X, et les valeurs de chaque colonne de la deuxième matrice comme les valeurs de l'axe Y.

Exemple :

```
>> K = [1 1 1;2 2 2; 3 3 3;4 4 4]
K =
     1     1     1
     2     2     2
     3     3     3
     4     4     4

>> M = [0 -2 1;2 0 3;-3 3 -2;1 1 4]
M =
     0    -2     1
     2     0     3
    -3     3    -2
     1     1     4
```



```
>>plot(K,M)
```

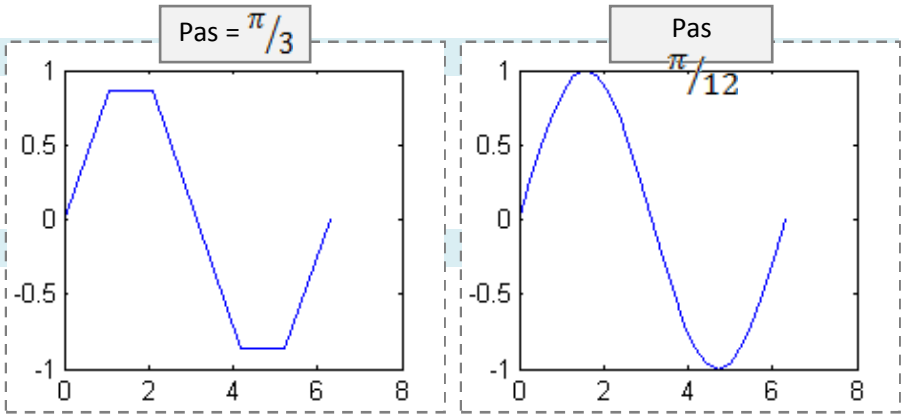
Il est évident que plus le nombre de coordonnées augmente plus la courbe devienne précise. Par exemple pour dessiner la courbe de la fonction $y = sin(x)$ sur [0, 2π] on peut écrire :

La première figure

```
>> x=0:pi/3:2*pi;
>> y = sin(x);
>>plot(x,y)
```

La deuxième figure

```
>> x=0:pi/12:2*pi;
>> y = sin(x);
>>plot(x,y)
```



6.2 MODIFIER L'APPARENCE D'UNE COURBE :

Il est possible de manipuler l'apparence d'une courbe en modifiant la couleur de la courbe, la forme des points de coordonnées et le type de ligne reliant les points.

Pour cela, on ajoute un nouveau argument (qu'on peut appeler un marqueur) de type chaine de caractère à la fonction ***plot*** comme ceci :

```
plot (x, y, 'marqueur')
```

Le contenu du marqueur est une combinaison d'un ensemble de caractères spéciaux rassemblés dans le tableau suivant :

Couleur de la courbe	
le caractère	***son effet***
b ou**blue**	courbe en bleu
g ou **green**	courbe en vert
r ou**red**	courbe en rouge
c ou **cyan**	entre le vert et le bleu
m ou **magenta**	en rouge violacé vif
y ou**yellow**	courbe en jaune
k ou **black**	courbe en noir

Style de la courbe	
le caractère	***son effet***
-	en ligne plein ———
:	en pointillé ·····
-.	en point tiret – · – ·
	– – –

Représentation des points	
le caractère	***son effet***
.	un point .
o	un cercle ●
x	le symbole **x**
+	le symbole **+**
*****	une étoile *
s	un carré ■
d	un losange ♦
v	triangle inferieur ▼
^	triangle supérieur ▲
<	triangle gauche ◄
>	triangle droit ►
p	pentagramme ★

--	en tiret	**h**	hexagramme ✶

Exemple :

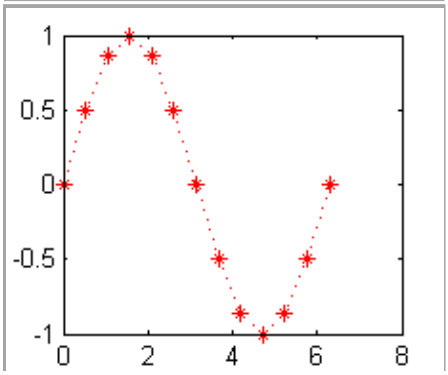
Essayons de dessiner la fonction *y=sin(x)* pour x = [0 ...2π] avec un pas = π/6.

```
>> x = 0:pi/6:2*pi;
>>y = sin(x);
```

En changeant le marqueur on obtient des résultats différents, et voici quelques exemples :

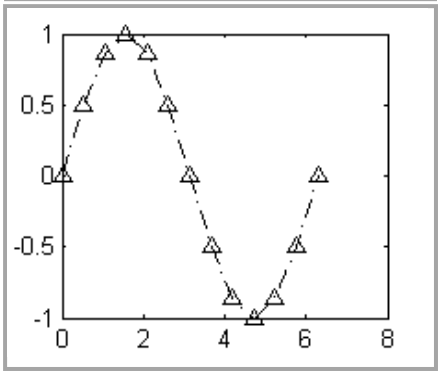
Couleur rouge, en pointillé et avec des étoiles

```
plot(x, y,'r:*')
```



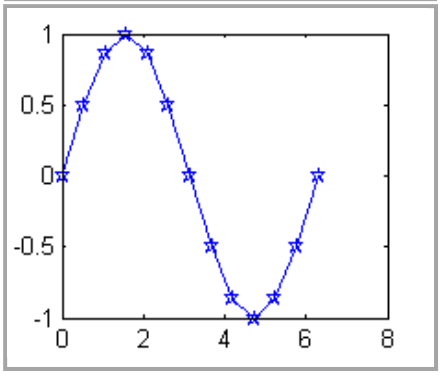
Couleur noire, en point tiret et avec des rectangles sup

```
plot(x, y,'black-.^')
```



Couleur bleu, en ligne plein et avec des pentagrammes

```
plot(x, y,'pb-')
```



6.3 ANNOTATION D'UNE FIGURE :

Dans une figure, il est préférable de mettre une description textuelle aidant l'utilisateur à comprendre la signification des axes et de connaitre le but ou l'intérêt de la visualisation concernée.

Il est très intéressant également de pouvoir signaler des emplacements ou des points significatifs dans une figure par un commentaire signalant leurs importances.

- ✓ Pour donner un titre à une figure contenant unecourbe nous utilisons la fonction ***title*** comme ceci :

```
>>title('titre de la figure')
```

- ✓ Pour donner un titre pour l'axe vertical des ordonnées y, nous utilisons la fonction ***ylabel*** comme ceci :

```
>>ylabel('Ceci est l''axe des ordonnées Y')
```

- ✓ Pour donner un titre pour l'axe horizontal des abscisses x, nous utilisons la fonction ***xlabel*** comme ceci :

```
>>xlabel('Ceci est l''axe des abscisses X')
```

- ✓ Pour écrire un texte (un message) sur la fenêtre graphique à une position indiquée par les coordonnées ***x*** et ***y***, nous utilisons la fonction ***text*** comme ceci :

```
>> text(x, y, 'Ce point est important')
```

- ✓ Pour mettre un texte sur une position choisie manuellement par la souris, nous utilisons la fonction ***gtext***, qui a la syntaxe suivante :

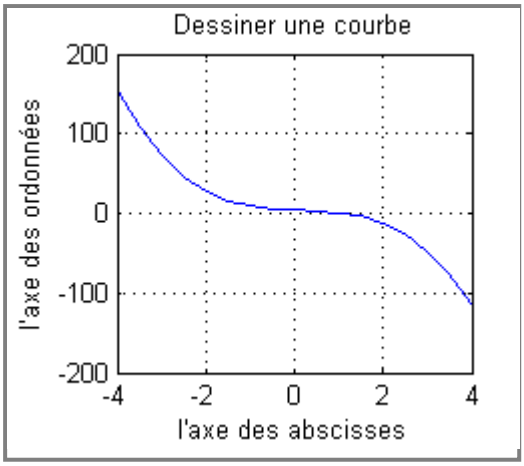
```
>>gtext('Ce point est choisi manuellement')
```

- ✓ Pour mettre un quadrillage (une grille), utilisez la commande ***grid*** (ou***grid on***). Pour l'enlever réutiliser la même commande **grid** (ou***grid off***).

Exemple :

Dessinons la fonction : $y=-2x^3+x^2-2x+4$ pour x varie de -4 jusqu'à 4, avec un pas = 0.5.

```
>> x = -4:0.5:4;
>> y = -2*x.^3+x.^2-2*x+4;
>> plot(x,y)
>>grid
>>title('Dessiner une courbe')
>>xlabel('l''axe des abscisses')
>>ylabel('l''axe des ordonnées')
```



6.4 DESSINER PLUSIEURS COURBES DANS LA MEME FIGURE :

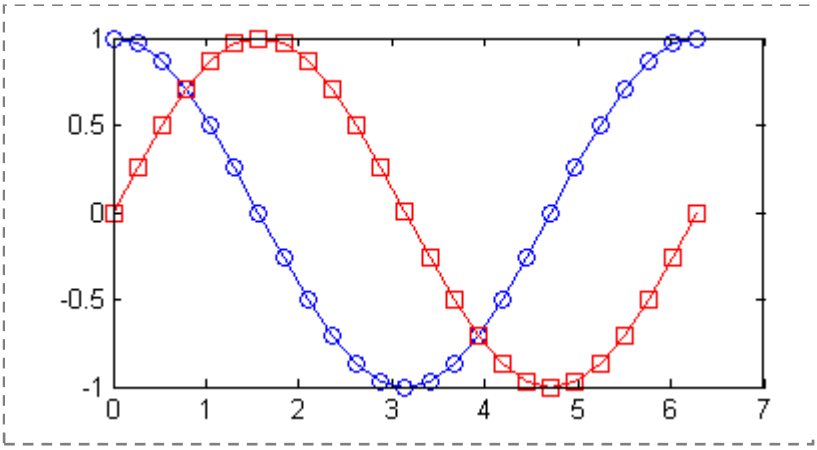
Par défaut en Matlab, chaque nouveau dessin avec la commande ***plot*** efface le précédent. Pour forcer une nouvelle courbe à coexister avec les précédentes courbes, Il existe au moins trois méthodes :

6.4.1 La commande hold

La commande ***hold*** (ou ***holdon***) active le mode 'préservation des anciennes courbes' ce qui permette l'affichage de plusieurs courbes dans la même figure. Pour annuler son effet il suffit de réécrire ***hold*** (ou ***hold off***).

Par exemple pour dessiner la courbe des deux fonctions *cos(x)* et *sin(x)* dans la même figure, on peut écrire :

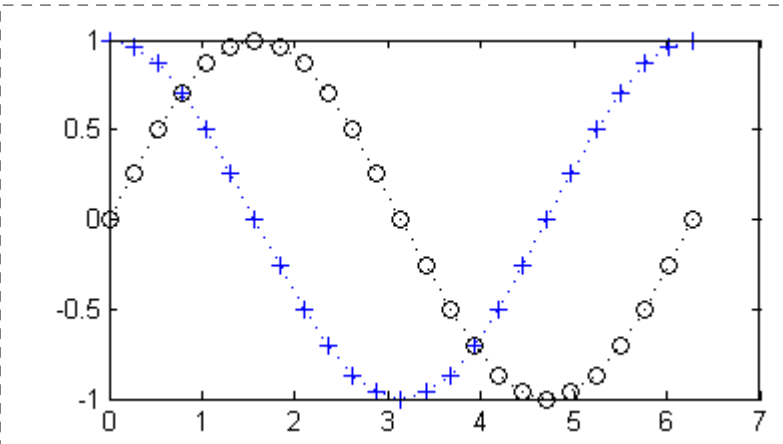
```
>> x=0:pi/12:2*pi;
>> y1=cos(x);
>> y2=sin(x);
>>plot(x,y1,'b-o')
>> hold on
>>plot(x,y2,'r-s')
```



6.4.2 Utiliser plot avec plusieurs arguments

On peut utiliser plot avec plusieurs couples (x,y) ou triples (x ,y, 'marqueur') comme arguments. Par exemple pour dessiner les mêmes fonctions précédentes on écrit :

```
>> x=0:pi/12:2*pi;
>> y1=cos(x);
>> y2=sin(x);
>>plot(x,y1,'b:+',x,y2,'k:o')
```



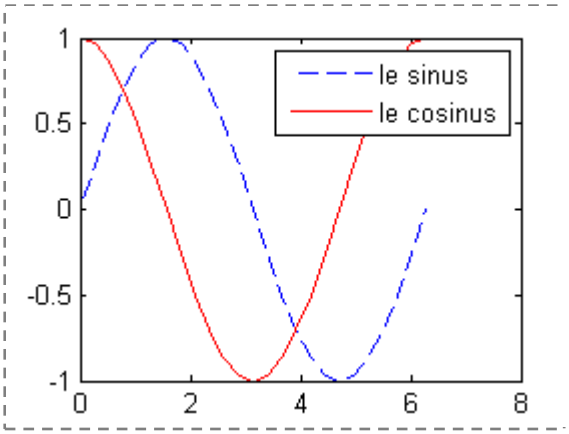
6.4.3 Utiliser des matrices comme argument de la fonction plot

Dans ce cas on obtient plusieurs courbes automatiquement pour chaque colonne (ou parfois les lignes) de la matrice. On a déjà présenté ce cas plus auparavant.

Après pouvoir parvenir à mettre plusieurs courbes dans la même figure, il est possible de les distinguer en mettant une légende indiquant les noms des ces courbes.

Pour cela, on utilise la fonction ***legend***, comme illustre l'exemple suivant qui dessine les courbes des deux fonctions *sin(x)* et *cos(x)* :

```
>> x=0:pi/12:2*pi;
>> y1=sin(x);
>> y2=cos(x);
>>plot(x,y1,'b--',x,y2,'-r')
>>legend('le sinus','le cosinus')
```



Il est possible de déplacer la légende (qui se situe par défaut dans le coin supérieur droit) en utilisant la sourie avec un glisser-déposer.

6.5 MANIPULATION DES AXES D'UNE FIGURE:

Matlab calcule par défaut les limites (le minimum et le maximum) des axes X et Y et choisie automatiquement le partitionnement adéquat. Mais il est possible de contrôler l'aspect des axes via la commande ***axis***.

Pour définir les limites des axes il est possible d'utiliser cette commande avec la syntaxe suivante :

axis ([xmin xmax ymin ymax]) Ou ***axis ([xmin,xmax,ymin,ymax])***

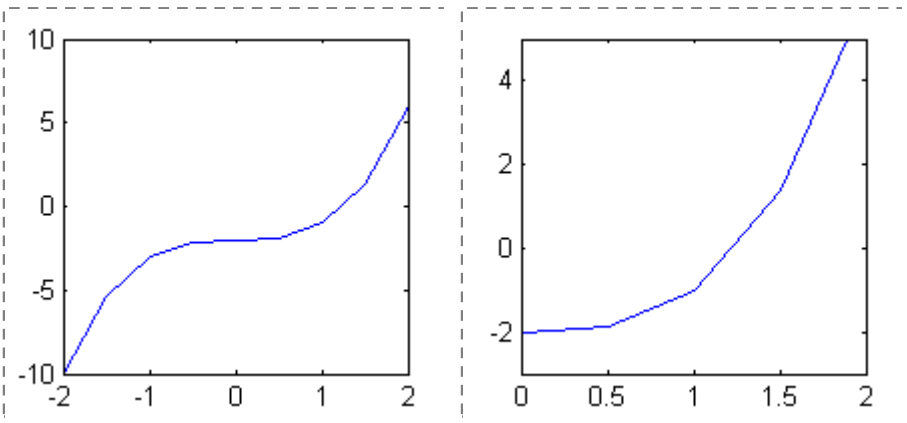
Avec : ***xmin*** et ***xmax*** définissent le minimum et le maximum pour l'axe des abscisses.
ymain et ***ymax*** définissent le minimum et le maximum pour l'axe des ordonnées.

Pour revenir au mode d'affichage par défaut, nous écrivons la commande : ***axis auto***

Exemple :

$f(x) = x^3-2$

```
>> x = -2:0.5:2;
>> y = x.^3-2;
>>plot(x,y)
```




```
>>axis auto
```

```
>> axis([0,2,-3,5])
```

Parmi les autres options de la commande ***axis***, nous présentons les suivantes :

- Pour rendre la taille des deux axes identique (la taille et non le partitionnement), nous utilisons la commande ***axis square***. Elle est nommée ainsi car elle rend l'aspect des axes tel un carré.
- Pour rendre le partitionnement des deux axes identique nous utilisons la commande ***axis equal***.
- Pour revenir à l'affichage par défaut et annuler les modifications nous utilisons la commande ***axis auto***.
- Pour rendre les axes invisibles nous utilisons la commande ***axis off***. Pour les rendre visibles à nouveau nous utilisons la commande ***axis on***.

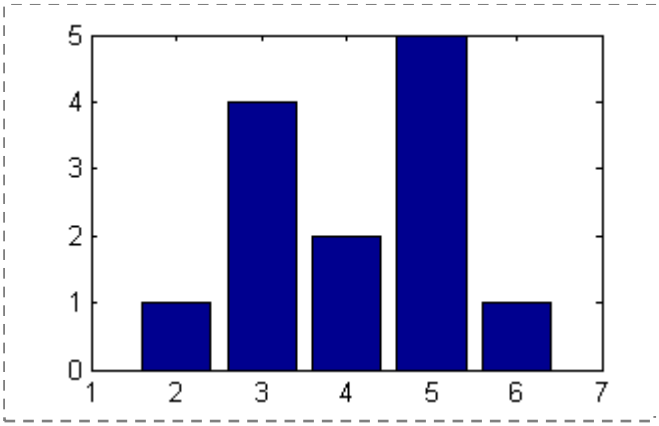
6.6 D'AUTRES TYPES DE GRAPHIQUES:

Le langage Matlab ne permet pas uniquement l'affichage des points pour tracer des courbes, mais il offre aussi la possibilité de tracer des graphes à bâtons et des histogrammes.

Pour tracer un graphe à bâtons nous utilisons la fonction ***bar*** qui a le même principe de fonctionnement que la fonction **plot**.

Exemple :

```
>> X=[2,3,5,4,6];
>> Y=[1,4,5,2,1];
>> bar(X,Y)
```



Il est possible de modifier l'apparence des bâtons, et il existe la fonction ***barh*** qui dessine les bâtons horizontalement, et la fonction ***bar3*** qui ajoute un effet 3D.

Parmi les fonctions de dessins très intéressantes non présentées (faute de place) on peut trouver : ***hist***, ***stairs***, ***stem***, ***pie***, ***pie3***, ...etc. (que nous vous encourageons à les étudier).

Nous signalons aussi que Matlab permet l'utilisation d'un système de coordonnées autre que le système cartésien comme le système de coordonnées polaire (pour plus de détail chercher les fonctions ***compass***, ***polar*** et ***rose***).

6.7 EXERCICES:

Exercice 01 :

Tracer la fonction $y1 = x * \log(x+1)$ et la fonction$y2 = 5e^{-0.2x} cos^2 x$ sur le même graphe.

Exercice 02 :

Tracer la courbe paramétrée définie dans le plan *xoy* par :

$$\begin{cases} x = t^2 - 3t \\ y = t^3 - 9t \end{cases} \quad pour - 4 \leq t \leq 4$$

Exercice 03 :

Tracer sur le même figure mais sur trois graphes différents les fonctions suivantes définies en coordonnées polaires :

1. $r = 1 + ecos\theta$
2. $r = \frac{1}{1+ecos\theta}$
3. $r = \sin(2\theta)\cos(2\theta)$

On prendra e=0.5

CHAPITRE 7

PRISE EN MAIN DE SIMULINK

7 Chapitre 07 : Prise en main de simulink

SIMULINK est un langage de programmation graphique qui permet de modéliser et de simuler des systèmes dynamiques.

7.1 QUELQUES BIBLIOTHEQUES *:*

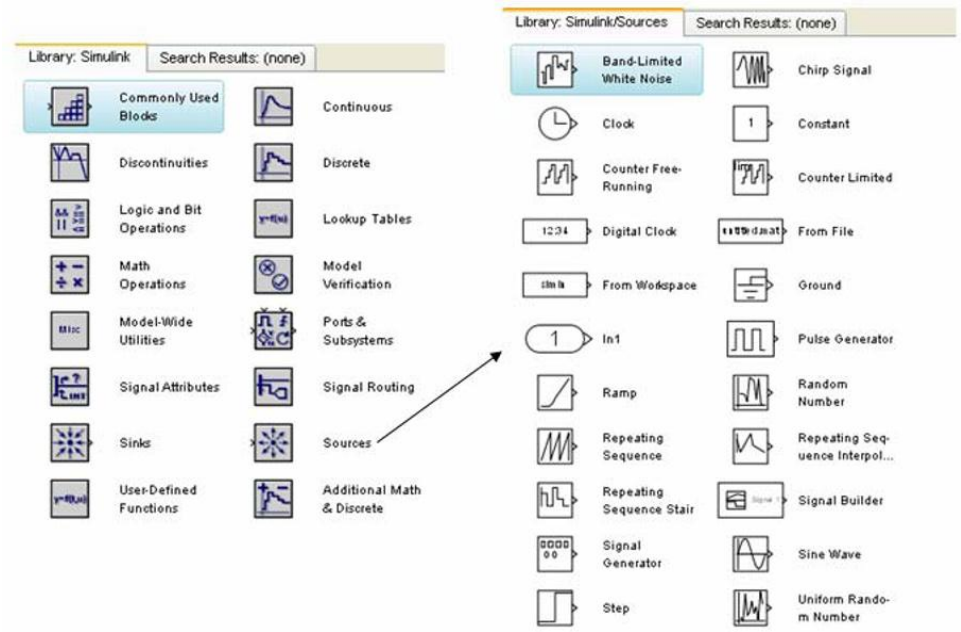
SIMULINK s'ouvre lorsqu'on clique sur le bouton 

Les librairies sont des ensembles de blocs répartis selon la catégorie de fonctions réalisées.

Parmi ces librairies, nous trouvons, entre autres, les plus utilisées :

Sources	Générateurs de signaux, lecture dans fichiers de données.
Sinks	Blocs d'affichage, enregistrement dans fichiers de données.
Signal Routing	Routage des fils de liaison entre blocs.
Math Operations	Operations mathématiques.
Continuous	Blocs continus.
Discrete	Blocs discrets.
Ports & Subsystems	Blocs permettant de réaliser des sous-systèmes.
Logic and Bit Operations	Blocs d'opérations logiques et binaires.
Discontinuities	Blocs discontinus (hystérésis, seuil, etc.).
Additional Math & Discrete	Blocs additionnels d'opérations mathématiques et de systèmes discrets.

Ci-dessous, nous présentons la librairie **Sources**.



Nous avons, dans cette bibliothèque, des générateurs de signaux, des blocs de lecture de fichiers textes, binaires, des variables de l'espace de travail, un bloc constant.

7.2 QUELQUES EXEMPLES *:*

7.2.1 Réponse indicielle d'un système du 1er ordre

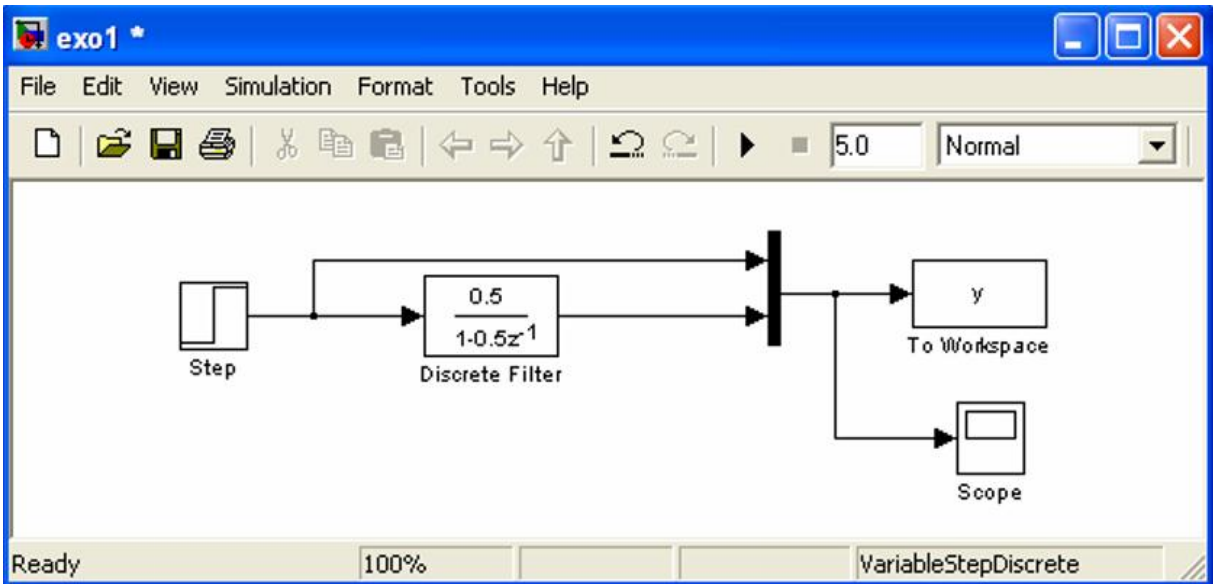
Le modèle exo1.mdl permet de simuler la réponse indicielle d'un système numérique du premier ordre de pôle 0.5 et de gain statique unité.

$$H(z) = 0.5/(1 - 0.5\ z^{-1})$$

Pour cela, nous utilisons le bloc générateur d'échelon de la librairie Sources que l'on relie à l'entrée du système (librairie Discrete).

La sortie du système ainsi que la commande échelon sont sauvegardées, grâce à un multiplexeur (librairie Signal Routing), dans la variable y que l'on peut utiliser dans l'espace de travail MATLAB.

Elle est de même affichée dans un oscilloscope (librairie Sinks).



En double-cliquant sur les blocs, on peut les paramétrer : édition du numérateur et dénominateur de la fonction de transfert, hauteur de l'échelon, ainsi que le type de variable y, structure ou tableau (Array)

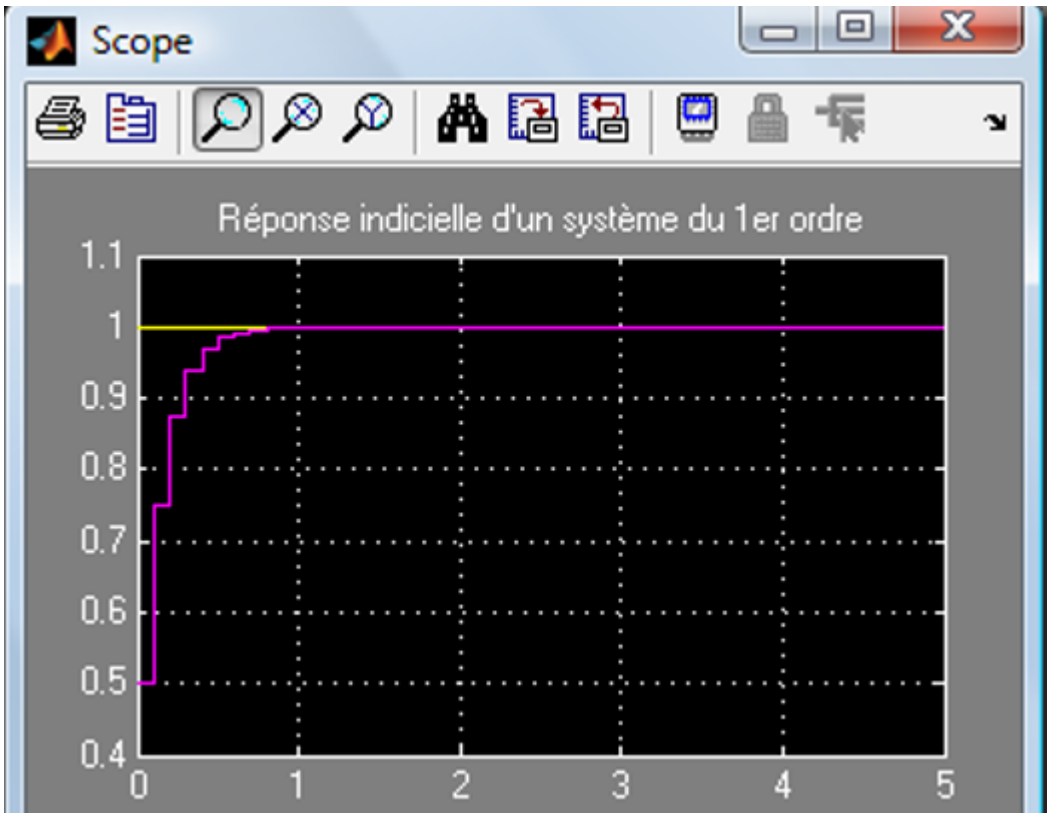
Avec l'option " Configuration Parameters " du menu Simulation, nous pouvons spécifier les paramètres de la simulation : durée de la simulation,

algorithme de résolution.

Dans ce qui suit, nous allons étudier d'autres exemples qui nous permettront d'étudier d'autres fonctionnalités de SIMULINK.

La courbe suivante de l'oscilloscope, montre l'entrée et la sortie du système discret du premier ordre. Nous vérifions bien le gain statique unité obtenu pour z=1, soit :

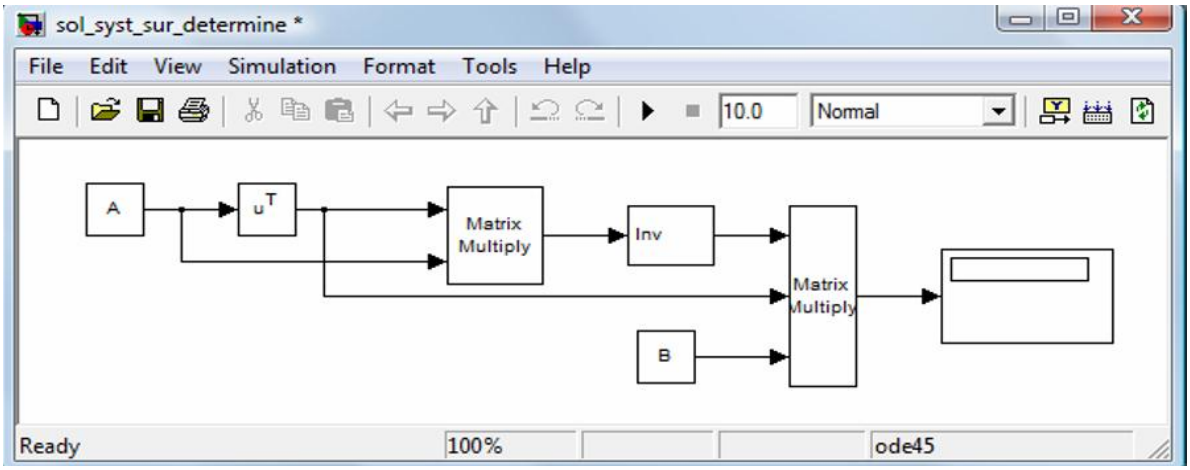
```
H(1) = 0.5/(1-0.5) = 1
```



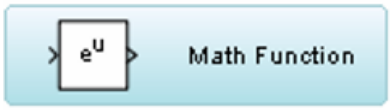
7.2.2 Résolution d'un système linéaire surdéterminé

Le modèle suivant permet de résoudre un système sur déterminé (ou déterminé) de la forme A x = B.

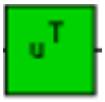
On programme la résolution d'un système sur déterminé par la méthode des moindres carrés :

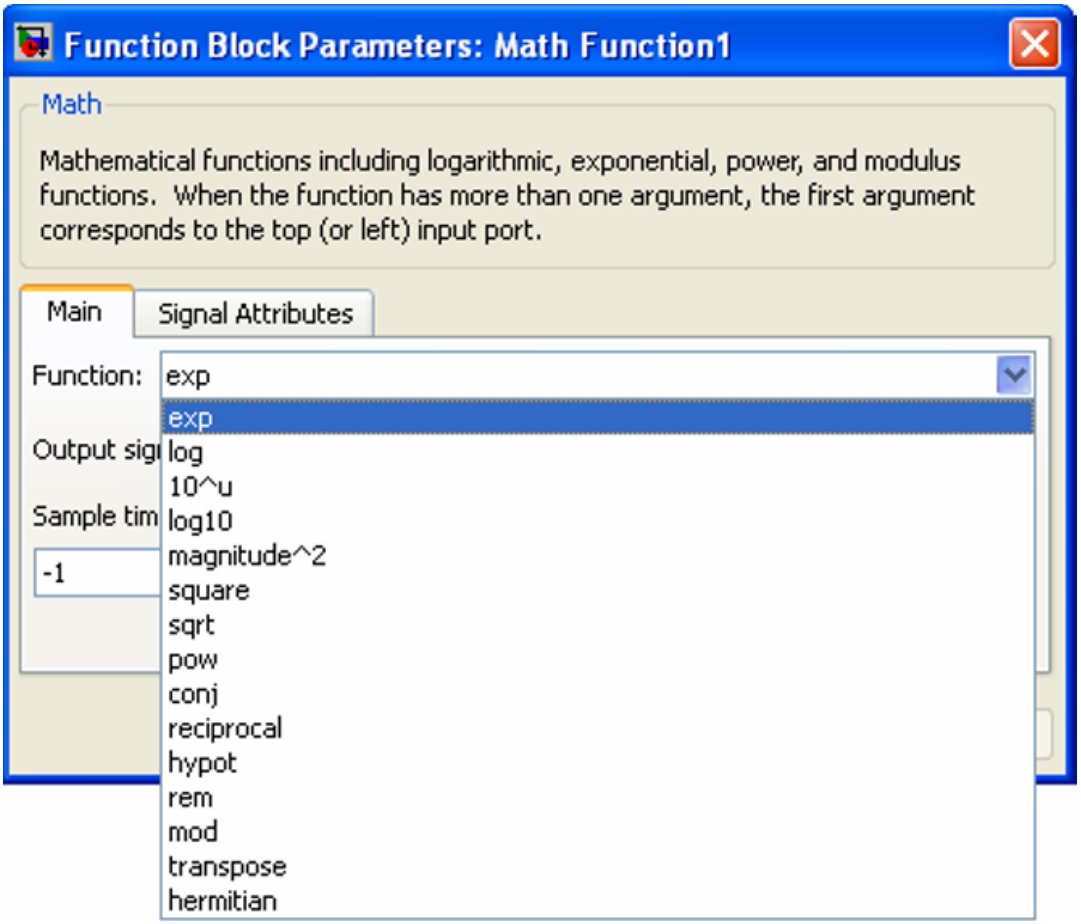


Pour la transposition, nous utilisons le bloc Math Function



de la librairie Math Operations.

Dans le menu déroulant Function on choisit l'option transpose. Le bloc prend alors la forme 



Quant à la multiplication, l'inversion ou le produit matriciel, nous utilisons le même bloc Product (produit) 

que nous paramétrons sous sa forme matricielle avec autant d'entrées que l'on désire.

On peut réaliser l'inversion par la division matricielle en écrivant le symbole « / » dans la case Number of inputs. Ainsi le bloc ne possèdera qu'une entrée (la matrice à inverser) sur laquelle est notée Inv.

La matrice A et le vecteur B sont entrés comme des constantes qui prennent les valeurs des variables spécifiées et connues dans l'espace de travail MATLAB.

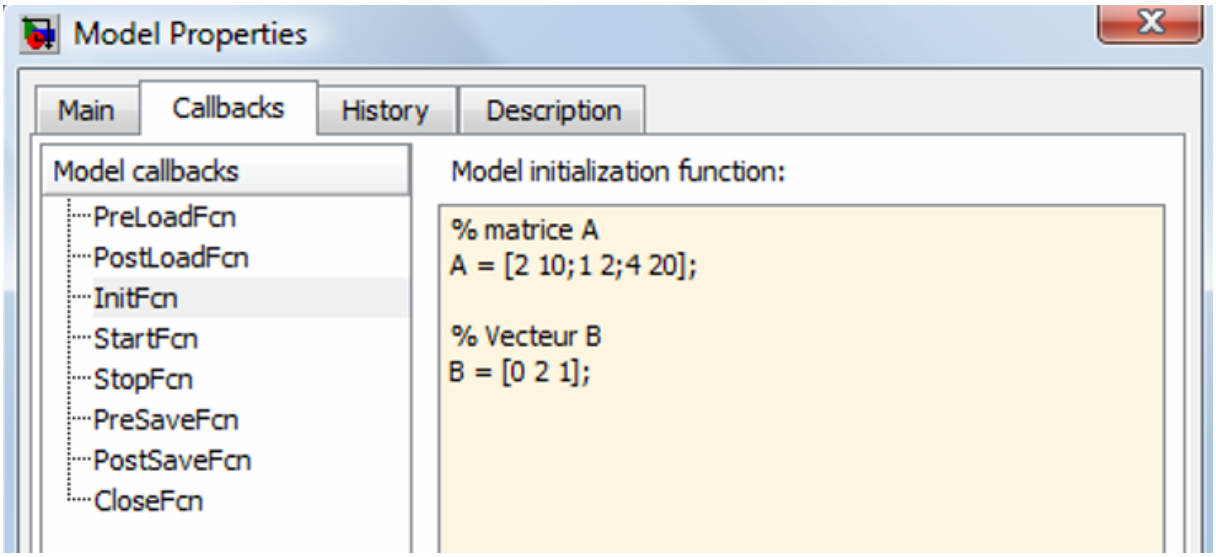
Considérons le système sur déterminé suivant :

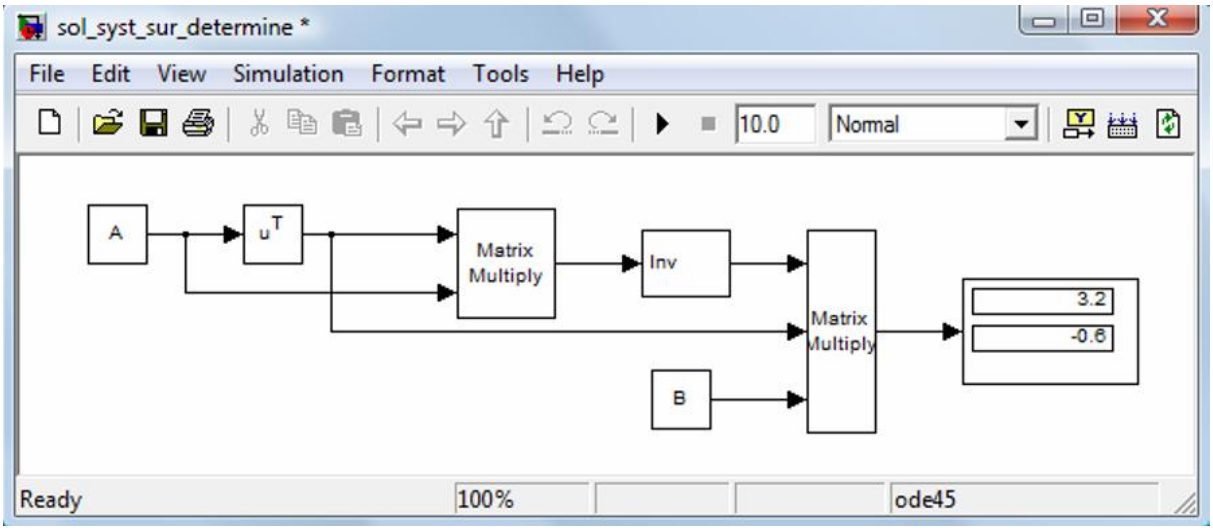
$$\begin{aligned} 2x_1+10x_2&=0\\ x_1+2x_2&=2\\ 4x_1+20x_2&=1 \end{aligned} \qquad \begin{vmatrix} 2 & 10\\ 1 & 2\\ 4 & 20 \end{vmatrix} \begin{bmatrix} x_1\\ x_2 \end{bmatrix} = \begin{vmatrix} 0\\ 2\\ 1 \end{vmatrix} \Leftrightarrow AX=B$$

```
>> A=[2 10;1 2;4 20];
>> B = [0 2 1]';
>> inv(A'*A)*A'*B
```

```
ans =
    3.2000
   -0.6000
```

Les matrices A et B, si elles ne sont pas connues dans l'espace de travail, peuvent être spécifiées dans un callback (« Callbacks »), tel le callback InitFcn.



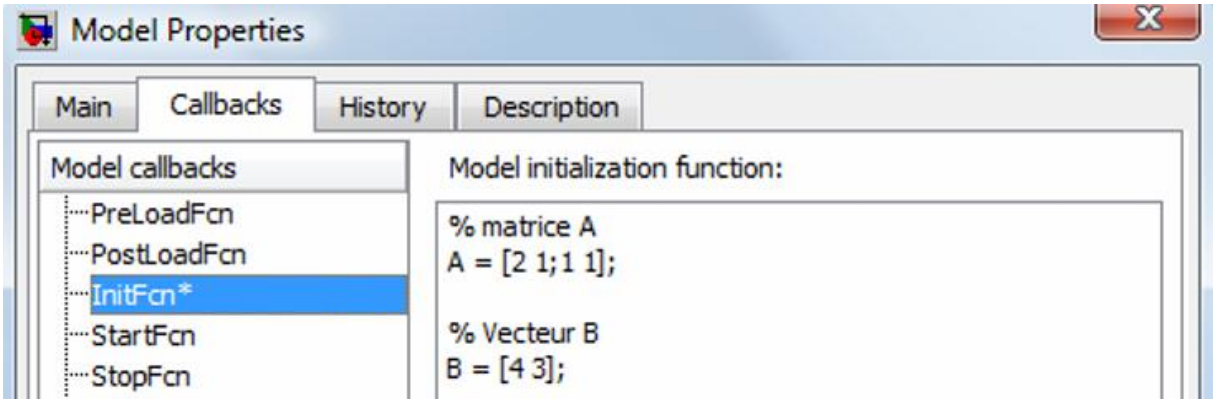


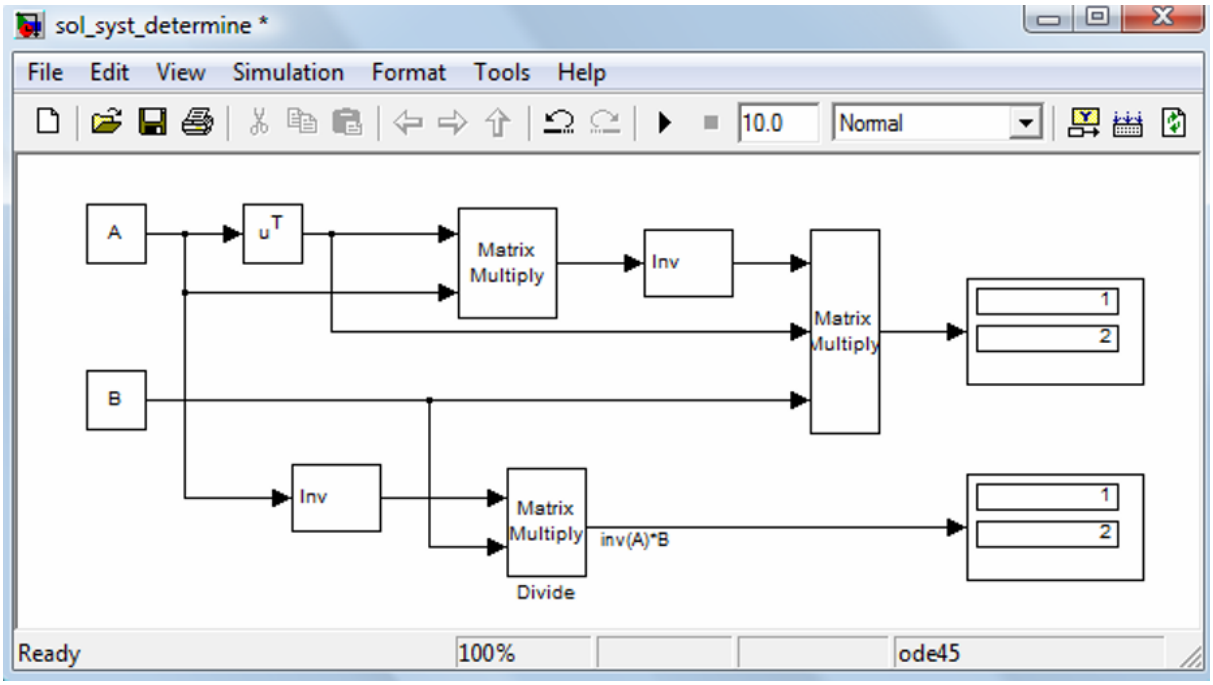
Dans le cas déterminé, de 2 équations à 2 inconnues, nous vérifions la solution suivante :

$$\begin{aligned} 2x_1+3x_2&=8\\ x_1-2x_2&=-3 \end{aligned} \qquad \begin{vmatrix} 2 & 3\\ 1 & -2 \end{vmatrix}\begin{bmatrix} x_1\\ x_2 \end{bmatrix}=\begin{bmatrix} 8\\ -3 \end{bmatrix}\Leftrightarrow AX=B$$

```
>> A = [2 1; 1 1];
>> B= [4 3]';
>> inv(A)*B
ans =
     1
     2
```

La matrice A et le vecteur B sont spécifiés dans le callback InitFcn.





7.2.3 Solution d'équation différentielle du 2nd ordre

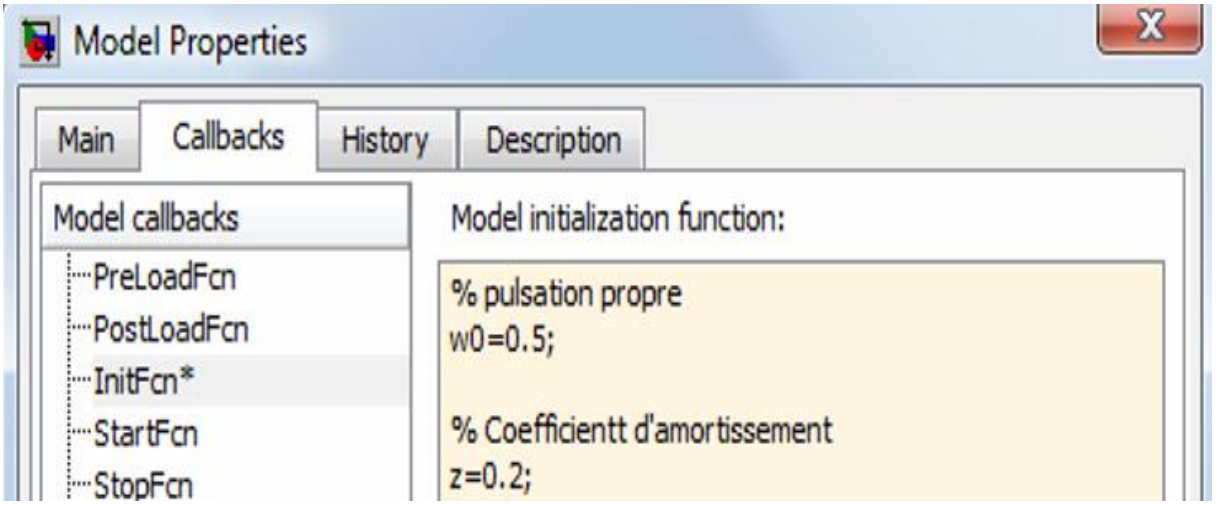
On se propose de trouver la solution de l'équation différentielle du 2nd ordre suivante :

$$s'' = w_0^2 (u - s) - 2\zeta w_0 s'$$

En prenant la transformée de Laplace, on remarque que cette solution est la sortie du système du 2nd ordre de fonction de transfert :

$$S(p)=\frac{1}{1+\frac{2\zeta}{w_0}p+\frac{p^2}{w_0^2}}U(p)$$

Le système est programmé ci-dessous sous la forme de son équation différentielle du second degré. La pulsation propre w_o et le coefficient d'amortissement sont spécifiés dans le callback InitFcn (étape d'initialisation du modèle SIMULINK« syst_2nd_ordre_equa_diff.mdl ».). Le signal d'entrée du système est formé d'un échelon unité jusqu'à l'instant 100 puis on le suit par une rampe de pente 0.05 et de valeur initiale nulle. Afin d'éviter le croisement des fils, nous avons envoyé le signal d'entrée dans une étiquette Goto nommée [u] que l'on récupère plus loin par l'étiquette From de même nom. Les valeurs de w_o et de ζ sont les suivants :

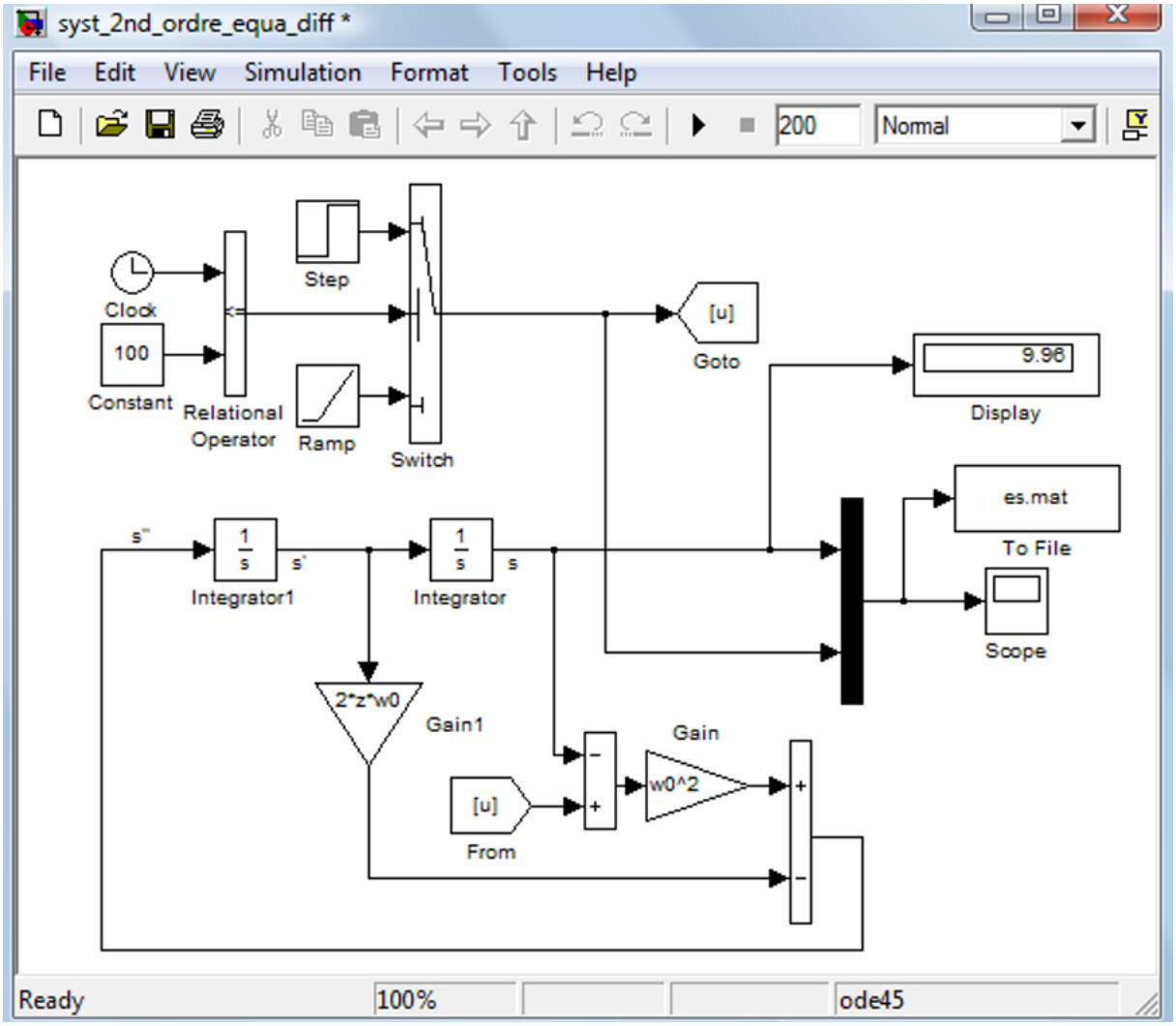


Nous appliquons le même signal d'entrée au système analogique du 2nd ordre et nous obtenons la même réponse temporelle.

Le signal d'entrée est un échelon suivi d'une rampe selon la condition spécifiée par l'opérateur relationnel (Relational Operator) de la bibliothèque Logic et Bit Operations.

L'échelon passe à travers le Switch lorsque la sortie de l'opérateur « <= » est au niveau logique 1 soit le temps d'horloge inférieur à 100. Le signal d'entrée est

ensuite envoyé dans la variable locale u, grâce au bloc Goto

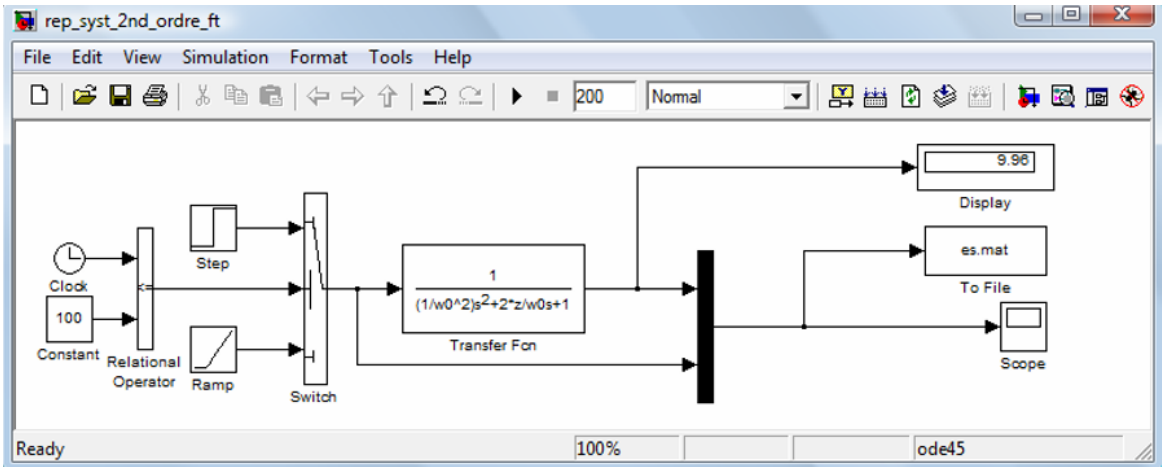


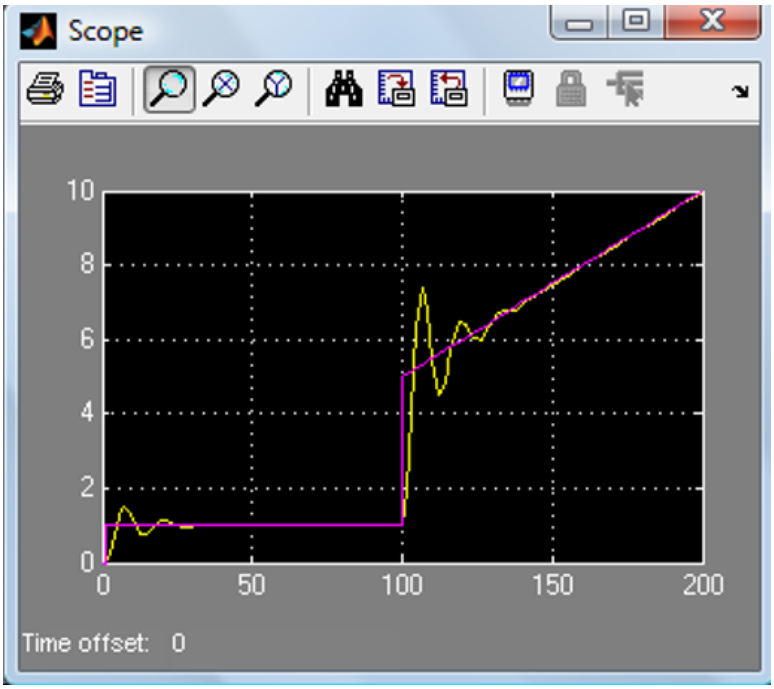
Sa valeur sera récupérée via la même variable u grâce au bloc From. Goto et From appartiennent à la bibliothèque Signal Routing.

L'équation différentielle est programmée par 2 intégrateurs purs et 2 gains dans lesquels on utilise les valeurs du coefficient d'amortissement ζ et la pulsation propre w_0 .

Les signaux d'entrée et de sortie sont envoyés à la fois vers l'oscilloscope, le fichier binaire es.mat et l'afficheur numérique Display.

Dans le modèle suivant, le même signal d'entrée est envoyé à l'entrée du système du 2nd ordre de coefficient d'amortissementζ , de pulsation proprew_0 et un gain statique unité.





Nous avons aussi sauvegardé ce signal dans le fichier binaire es.mat que l'on peut lire grâce à la commande load.

On efface toutes les variables de l'espace de travail et on lit le fichier binaire.

```
>> clear all
>> load es
```

La seule variable de l'espace de travail est celle portant le nom du fichier binaire.

```
>> who
Your variables are:
es
```

Bien que le multiplexeur possède uniquement 2 entrées, le signal enregistré possède 3 lignes et autant de colonnes que le nombre d'échantillons temporels.

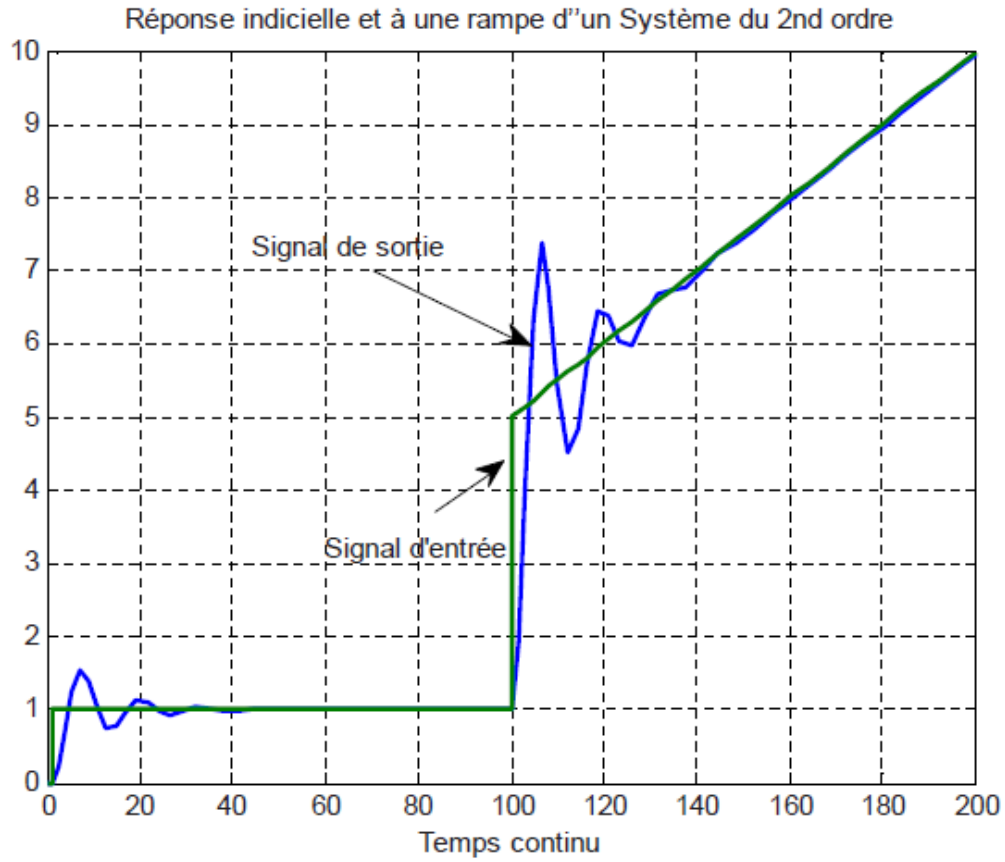
```
>> size(es)
ans =
     3    84
```

Par défaut, MATLAB ajoute toujours la variable temps en début de fichier, soit le vecteur es(1, :) dans notre cas ici.

```
>> plot(es(1,:), es(2,:), es(1,:), es(3,:))
>> title('Réponse indicielle et à une rampe d''un Système …
du 2nd ordre')
>> xlabel('Temps continu')
>> gtext('Signal de sortie')
>> gtext('Signal d''entrée')
```

Le bloc numérique display de la bibliothèque sinks affiche les valeurs instantanées du signal de sortie du système du 2nd ordre analogique.

Nous obtenons les mêmes signaux que ceux affichés sur l'oscilloscope.



Après quelques oscillations du fait du faible coefficient d'amortissement, le signal

de sortie rejoint le signal d'entrée car le gain statique est égal à 1.

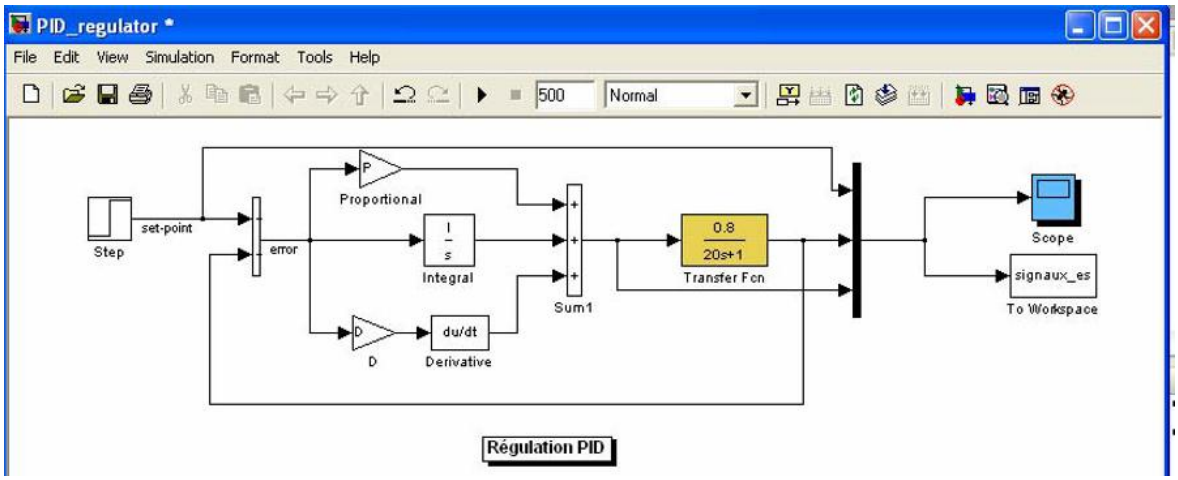
7.2.4 Régulation PID

On se propose de réaliser une régulation PID du processus du 1er ordre de fonction de transfert : $H(p)=\frac{0.8}{1+20\,p}$

Le système à contrôler est du premier ordre de constante de temps τ =20 s et de gain statique égal à 0.8.

Le PID est réalisé sous forme de la somme des 3 actions :

- Proportionnelle, P
- Intégrale, $\frac{1}{p}$ avec un gain I.
- Dérivée, $\frac{du}{dt}$, précédée du gain D.



Le gain Dérivée est choisi égal à 0, le régulateur a pour expression :

$$R(p)=0.1+\frac{0.05}{p}$$

```
>> P=0.1;
>> I=0.05;
>> D=0;
>> signaux_es

signaux_es =
         time: []
      signals: [1x1 struct]
    blockName: 'PID_regulator/To Workspace'
```

Les signaux sont sauvegardés dans la structure nommée signaux_es.

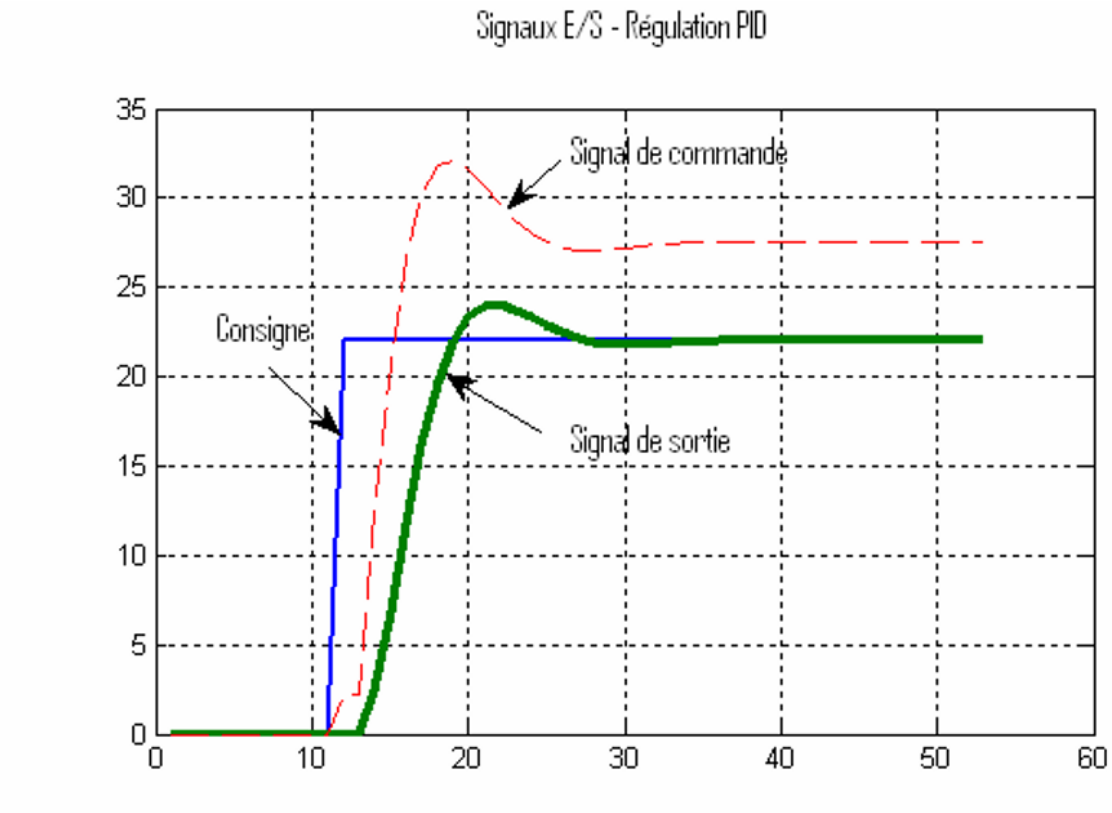
```
>> isstruct(signaux_es)

ans =
     1
```

La commande isfield retourne 3 valeurs logiques vraies, signaux_es est bien une structure à 3 champs : time, signals, block Name.

```
>>f=isfield(signaux_es,{'time','signals','blockName'})

f =
     1     1     1
```

Les 3 valeurs 1 (vraies) de la réponse f de la commande isfield indiquent que time, signals et block Name sont des champs de la structure signaux_es.

```
>> plot(signaux_es.signals.values)
```



En régime permanent le signal de sortie suit parfaitement le signal de consigne. Nous observons ces mêmes signaux sur l'oscilloscope.

CHAPITRE 8

Simulation des Machines A Courant Continu dans l'environnement Matlab/Simulink

8 Chapitre 08 : Simulation des Machines A Courant Continu dans l'environnement Matlab/Simulink

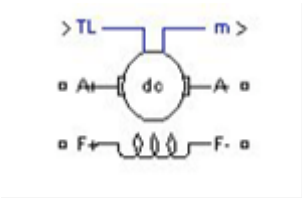
8.1 Introduction

Il s'agit, dans ce TP, de faire la modélisation et la simulation de systèmes électromécaniques comme les machines à courant. Pour ce faire, nous utiliserons le logiciel MATLAB auquel est intégré l'outil SIMULINK qui est une référence en la matière. Simulink est une plate-forme de simulation multi-domaine et de modélisation de systèmes dynamiques. Il fournit un environnement graphique et un ensemble de bibliothèques contenant des blocs de modélisation qui permettent le design précis, la simulation, l'implémentation et le contrôle de systèmes de communications et de traitement du signal

La finalité de la manipulation sera ainsi d'étudier le fonctionnement des différents types de machines à courant continu (**série, shunt, séparée**) en simulant puis en interprétant leurs courbes de fonctionnement.

8.2 Principe

Dans le cadre de la modélisation, la machine à courant à continu prend la forme d'un masque représenté par le bloc simplifié qui suit



Les circuits de l'induit et d'excitation (inducteur) sont visibles à partir du bloc DC Machine (A et F). A l'entrée TL, on applique le couple de la charge, la sortie m est destinée pour la mesure et l'observation des variables d'état de la machine dans l'ordre suivant : la vitesse angulaire, le courant dans l'induit, le couple électromagnétique.

Pour configurer la machine en machine série, shunt ou séparée, il suffira de changer les schémas de connexions entre l'induit et l'inducteur.

8.3 Matériel utilisé

- Une machine à courant continu :**DC machine** (bibliothèque SimPower System Blockset/Machines) ;
- Deux sources de tension continue : **DC Voltage Source**(Sim Power System Blockset / Electrical Sources) ;
- Un bloc Moment pour fournir le couple de charge (blocs **Gain** ou **Step** de la bibliothèque Simulink / Sources) ;
- Un bloc oscillographe **Scope** pour visualiser les processus (de la bibliothèque Simulink / Sinks) ;
- Un bloc **Voltage Measurements**pour la mesure de tension du circuit (de la bibliothèque SimPowerSystems/Measurements)
- Un bloc **Demux** à 4 sorties pour avoir accès aux 4 paramètres de la MCC (disponible dans la bibliothèque Simulink/Commonly Used Blocks) et un bloc **Mux** à deux sorties.

8.4 Déroulement du TP

8.4.1 Simulation d'une machine à courant continu à excitation séparée

Pour faire la simulation, nous réalisons le modèle ci-dessous:

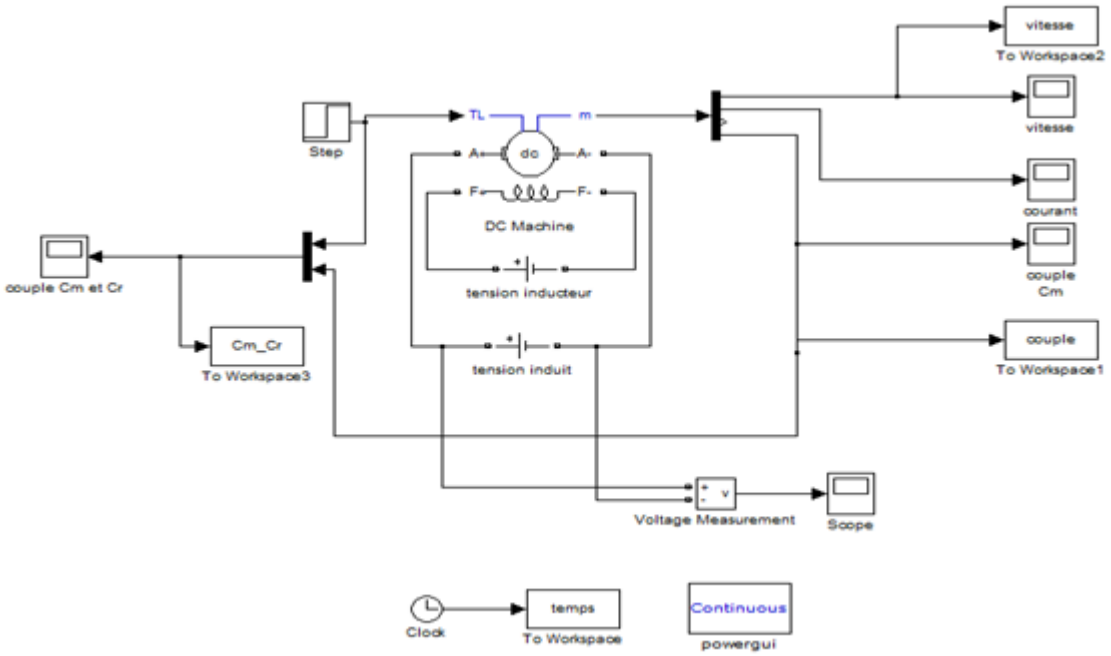


Figure 8.1 : Simulation d'une machine à courant continu à excitation séparé

Dans le champs de réglage de machine, on demande de régler les paramètres suivants :

- les paramètres de l'enroulement de l'induit : R_a=2,52 Ohm, L_a=0,048 H
- les paramètres de l'enroulement d'excitation R_f =92 Ohm, L_f=5,257 H, L_{af}=0,257 H
- la somme des moments d'inertie de la machine et de la charge : J=0,017 (Kgm2)
- le coefficient de frottement visqueux B_m=0,0000142 N.m.s ;
- le coefficient de frottement à sec : T_f =0,005968 Nm

On effectue enfin les derniers réglages sur le schéma ci-dessus en réduisant le temps de simulation de 10 à 0.5 seconde, en alimentant l'inducteur sous 220 V, l'induit sous 240V, et en réglant le couple résistant constant sur 10Nm.

On peut maintenant passer à la simulation et à la visualisation des courbes

8.4.1.1 Vitesse de rotation

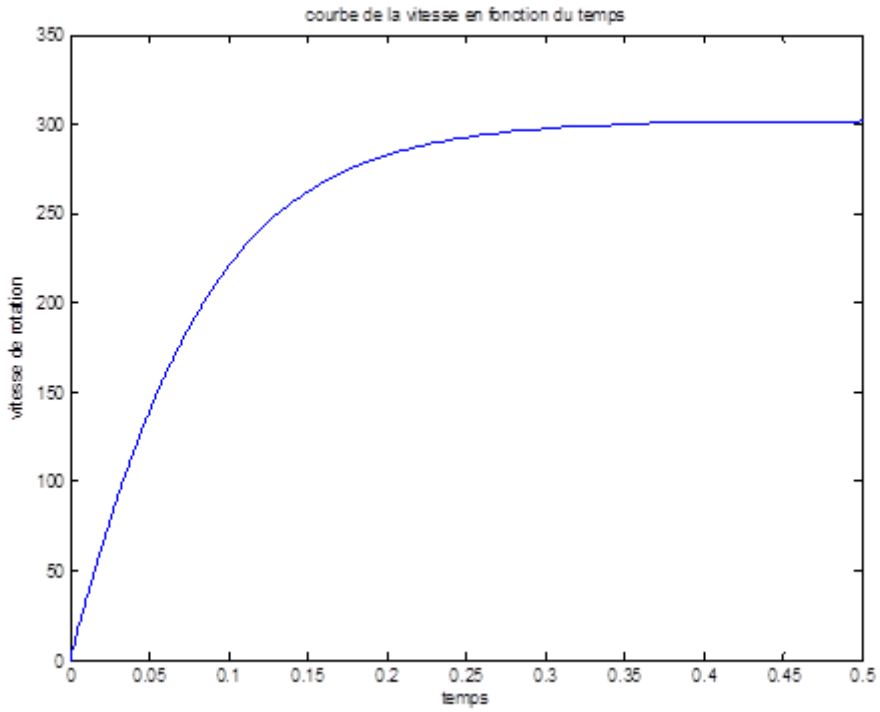


Figure 8.2 : courbe de la vitesse en fonction du temps

La vitesse part de zéro pour se stabiliser autour de 300 tr/min, qui est la vitesse en régime établi. La courbe a une forme exponentielle, ce qui correspond à

la présence de phénomènes transitoires dans les enroulements de la machine.

8.4.1.2 Couple moteur

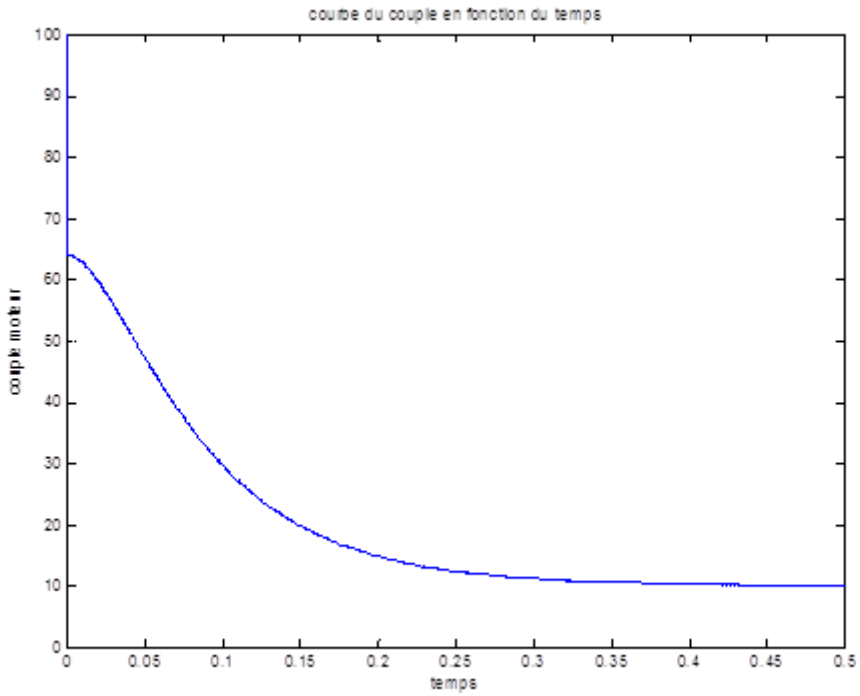


Figure 8.3 : Courbe du couple en fonction du temps

Le moteur démarre avec un couple de démarrage important de l'ordre de 65 N.m, puis le couple passe du régime transitoire au régime établi pour se stabiliser autour de 10 N.m, qui correspond au couple résistant imposé par la charge (TL).

8.4.1.3 Couple moteur et couple résistant

Le couple résistant reste constant et fixé à 10 N.m ; le couple moteur passe du régime transitoire au régime établi pour se stabiliser autour de 10 N.m. Les deux courbes se confondent autour de l'instant 0.5 seconde, et leur intersection définit le point de fonctionnement de la machine.

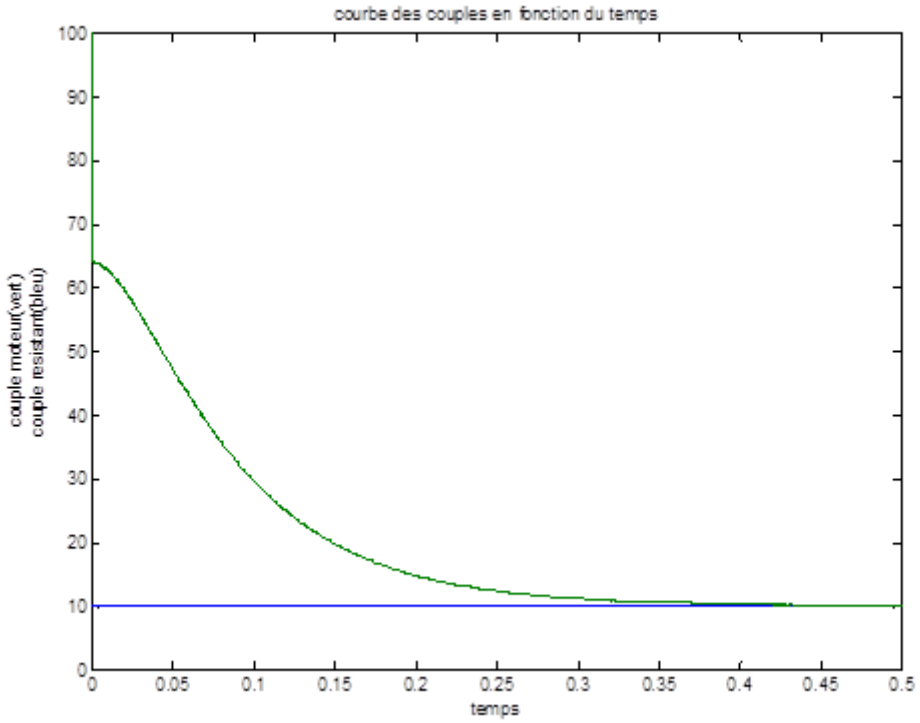


Figure 8.4 : Courbe des couples en fonction du temps

8.4.2 Simulation d'une machine à courant continu à excitation shunt

Dans cette partie, on utilise les mêmes outils que pour la simulation du moteur à excitation indépendante, sauf qu'ici la source de tension de l'inducteur est supprimée et l'inducteur est mis en parallèle sur la source de tension de l'induit suivant le modèle de la figure suivante.

L'induit est alimenté sous une tension de 240V et le couple résistant est nul. Le moteur tourne donc à vide

Cette machine à excitation shunt peut être assimilée à la machine à excitation indépendante précédente tournant à vide .En effet,la tension d'alimentation des enroulements ainsi que les paramètres internes du moteur restent les mêmes.

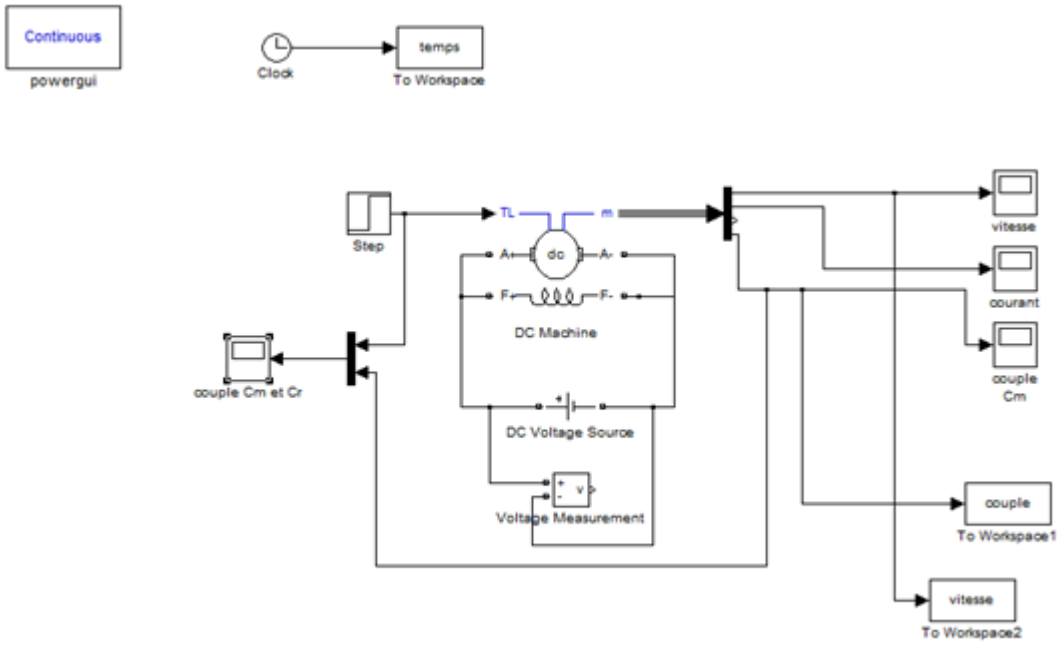


Figure 8.5 : Simulation d'une machine à courant continu à excitation shunt

. 8.4.2.1 Vitesse de rotation

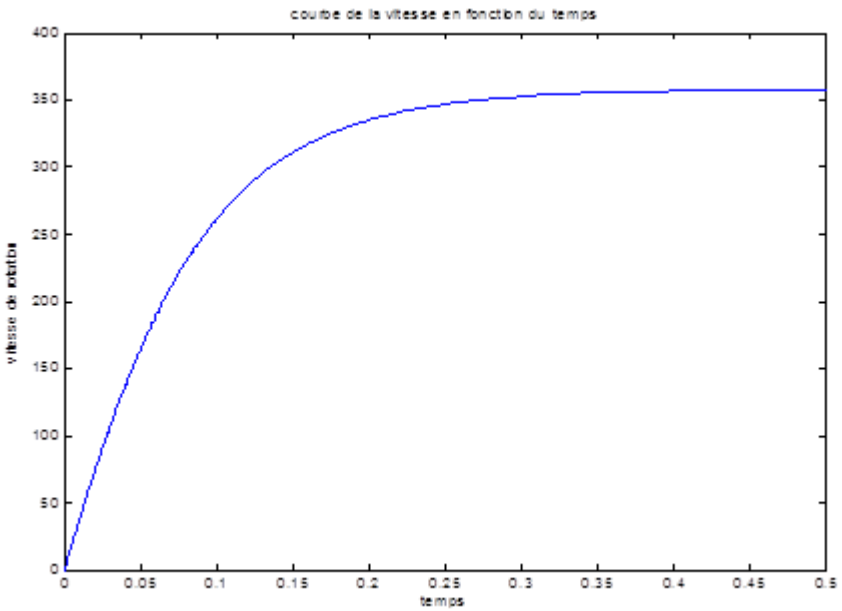


Figure 8.6 : courbe de la vitesse en fonction du temps

La vitesse augmente puis se stabilise autour de 360 tr/min.

On se rend compte ici que la vitesse en régime établi est plus grande ici que pour le moteur précédent (moteur excitation indépendante). Ici, en effet, c'est le

fait qu'il n'y ait pas de couple de charge et donc pas d'inertie supplémentaire qui fait tourner le moteur plus vite.

8.4.2.2 Couple moteur

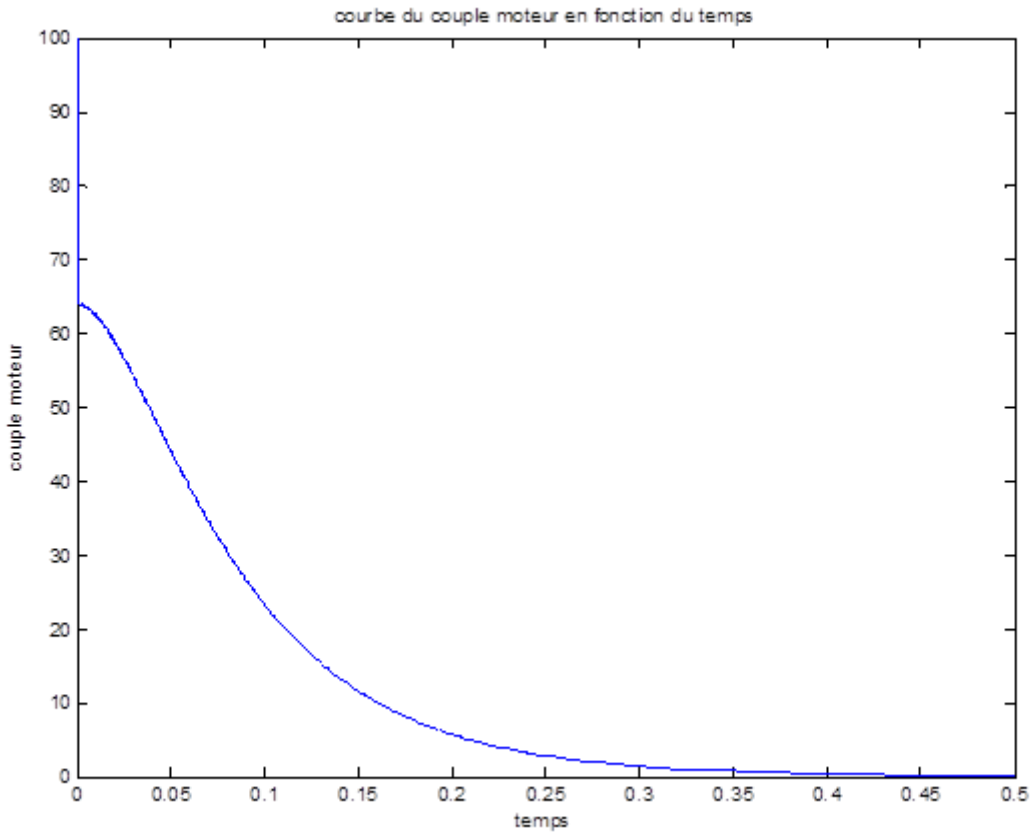


Figure 8.7 : Courbe du couple moteur en fonction du temps

Le couple moteur en régime établi est ici aussi plus petit que pour le moteur précédent (pratiquement nul); ce qui est normal puisqu'il n'y a pas de couple de charge à vaincre ici. En réalité, le couple moteur développé n'est pas nul, un petit couple est développé par le moteur pour vaincre le couple de pertes du aux pertes par frottements dans les parties mécaniques de la machine.

8.4.3 Simulation d'une machine à courant continu à excitation série

Dans cette partie, on garde les mêmes blocs utilisés précédemment à la seule exception du bloc « Gain » qui vient remplacer le bloc « Step » pour jouer le rôle de

charge mécanique. De plus l'inducteur est mis en série avec l'induit suivant le modèle de la figure ci-dessous

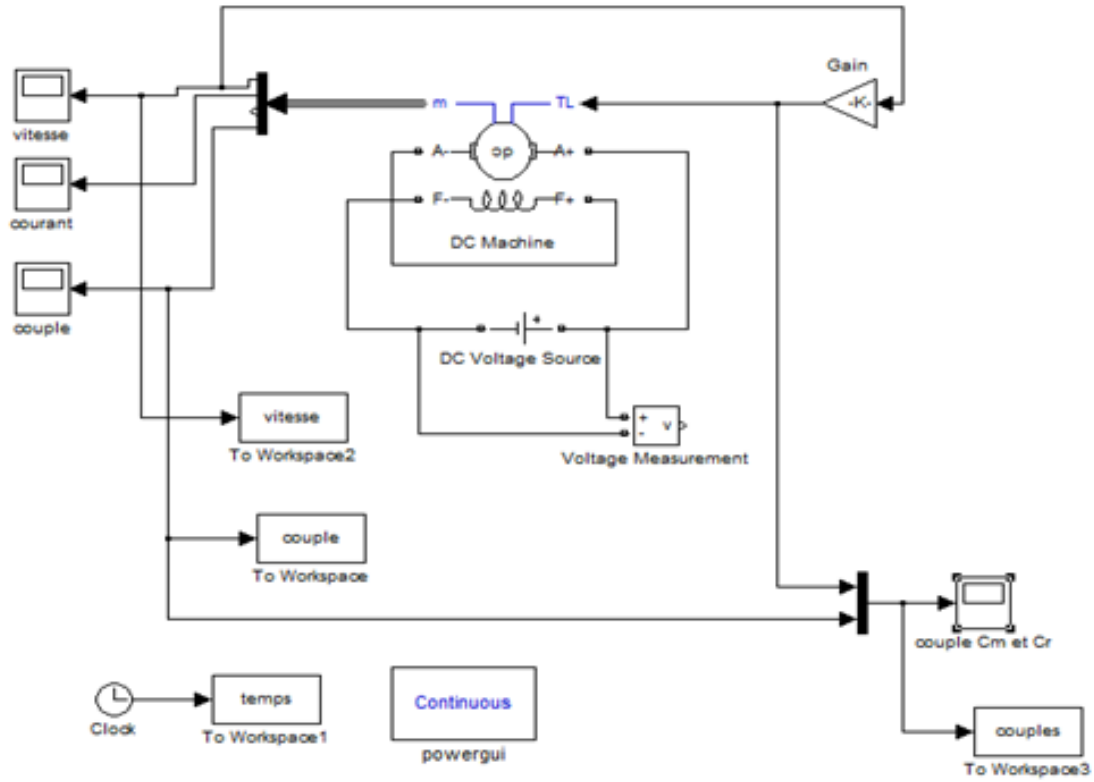


Figure 8.8 : Simulation d'une machine à courant continu à excitation série

On change également les paramètres inducteur suivant Rf= 2.581Ω, Lf=0.028H, Laf=0.09483H, J=2*0.02215 kg.m2 ,Bm= 2*0.02953, Va=240V.

En utilisant toujours le bloc DEMUX on visualise les courbes de variation de la vitesse, du courant d'induit et du couple, en fonction du temps en prenant le soin de prendre un couple résistant de type TL =kΩ.On utilise toujours la méthode d'importation des courbes dans l'espace de travail Matlab.

. 8.4.3.1 Vitesse de rotation

La vitesse augmente puis se stabilise en régime établi autour de 90 tr/min

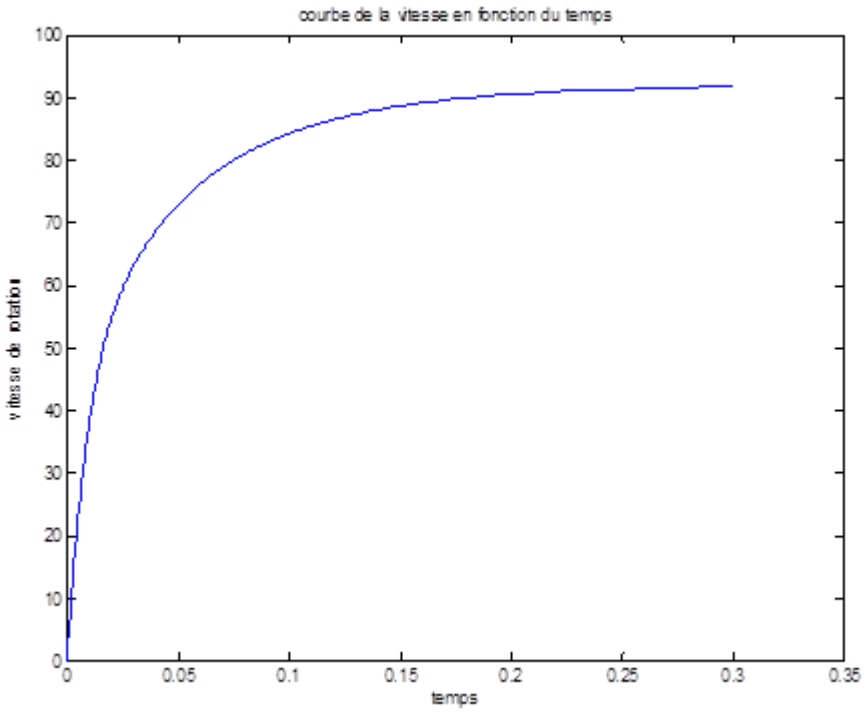


Figure 8.9 : courbe de la vitesse en fonction du temps

8.4.3.2 Couple moteur

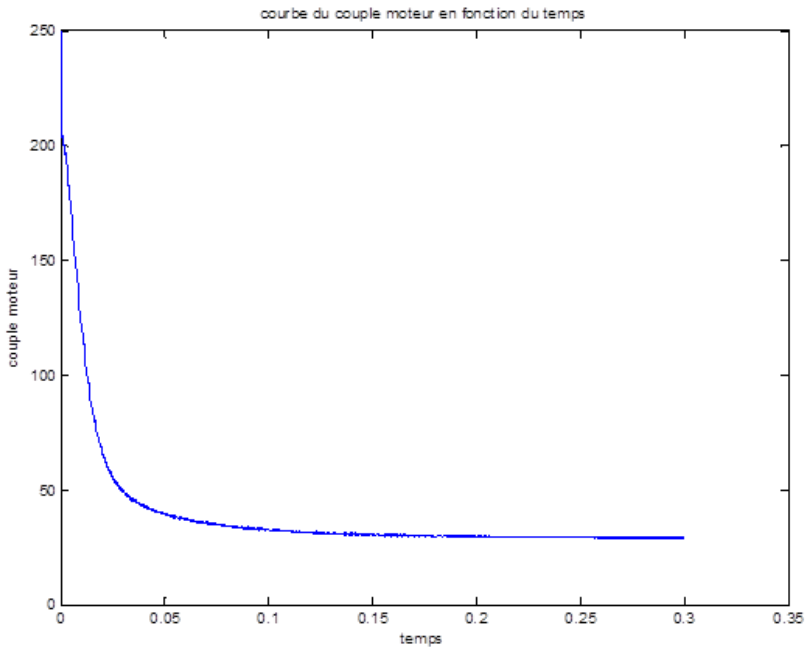


Figure 8.10: Courbe du couple moteur en fonction du temps

On remarque sur les courbes de la vitesse et du couple, que lorsque le couple augmente, la vitesse diminue.

8.4.3.3 Couple moteur et couple résistant

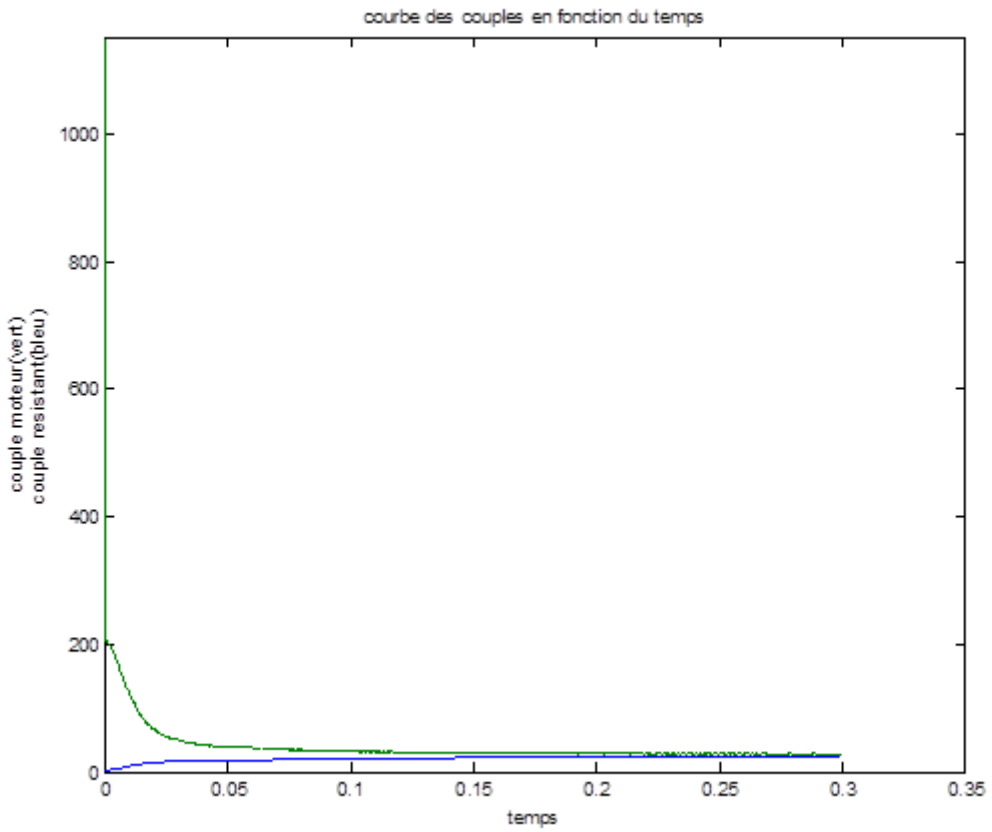


Figure 8.11 : Courbe des couples en fonction du temps

Puisque le couple résistant est de type gain (K*Ω), la courbe du couple résistant suit l'évolution de la courbe de la vitesse. La courbe de vitesse est croissante et se stabilise en régime permanent autour de 90 tr/min. La courbe de couple résistant est donc aussi croissante et se stabilise autour de0,25*90 c'est-à-dire autour de 22.5 N.m.

Quant au couple moteur, il est aussi régulé par la valeur du couple résistant en régime établi. En effet lorsque le couple résistant impose un certain courant dans l'induit, le même courant traverse l'inducteur et produit un flux proportionnel au courant, et donc un couple moteur proportionnel à l'appel de courant.

8.5 Conclusion

Ce TP nous a permis d'utiliser un laboratoire virtuel de simulation des machines à courant continus afin s'intéresser à leur étude.

Les résultats obtenus dans nos laboratoires virtuels nous montrent des caractéristiques qui sont en concordance avec celles obtenues avec les modèles théoriques. Nous en concluons que le modèle réalisé est assez fiable et précis.

Le logiciel MATLAB/SIMULINK, est un bon moyen d'étude du fonctionnement des machines à courant continus (et d'autres types de machines également) dans les conditions de fonctionnement voulues. Il nous permet d'observer de manière réaliste des phénomènes électriques et physiques(couple,vitesse, courant) dans et d'envisager des conditions de fonctionnement particulières.

CHAPITRE 9

Autres logiciels

9 Chapitre 09 : Les autres logiciels

9.1 HOMER :

9.1.1 Présentation du logiciel

Le logiciel HOMER (Hybrid Optimization Model for Electric Renewables [167, 168]) est un outil destiné à la simulation et l'optimisation de systèmes de génération électrique distribuée, que ce soit connectés à un réseau électrique classique ou bien isolés. Un de ses grands atouts est la possibilité de pouvoir simuler des systèmes hybrides combinant différentes sources d'énergie qu'elle soit renouvelable ou fossile.

Le modèle doit être renseigné des ressources disponibles, des options technologiques, des coûts et des charges à respecter par le système. Plusieurs composants peuvent être modélisés : des champs photovoltaïques, des éoliennes, des générateurs diesel, des batteries, des convertisseurs, etc.

En ce qui concerne les composants, des fonctions de coût linéaires sont adoptées et les dimensions des composants à prendre en considération doivent être préalablement (pré-dimensionnés) prévues afin de réaliser l'optimisation. Le nombre de tailles multiplié par le nombre de composants donne le nombre de configurations de systèmes simulés par le programme.

HOMER est un modèle de série chronologique, il effectue un bilan énergétique horaire le long d'une année pour chaque configuration de système rentrée par l'utilisateur. Il permet de faire la simulation d'un système selon des données de gisement (solaire, éolien, diesel, etc.) en fonction d'une demande énergétique (besoins en énergie). Par la suite, il affiche la liste des configurations de systèmes triés par le coût actuel net (NPC : Net Present Cost) qui combine le coût en capital annualisé par l'amortissement du composant au cours de sa durée de vie en utilisant les taux d'actualisation réels et celui de remplacement, fonctionnement et entretien.

Ensuite, il est finalement possible de faire des analyses de sensibilité afin de savoir si la solution trouvée reste la meilleure même s'il y a certains changements dans les différents paramètres d'entrées(variation du coût de la technologie, variation dans les données de gisement, etc.). Il est donc possible de faire bon

nombres d'analyses avec de nombreuses configurations différentes en moins de quelques minutes de simulation.

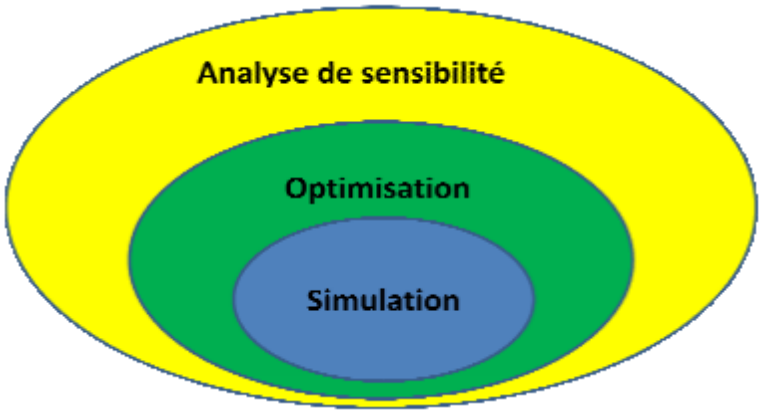


Figure 9.1: Modèle conceptuel du logiciel HOMER

HOMER est largement utilisé(par plus de 190 pays et plus de 40000 utilisateurs) et accepté comme un outil de simulation. La première version 1993 a subi de nombreuses améliorations dans plus de 40nouvelles versions.

9.1.2 Méthodologie

La mise en œuvre d'unités de production d'énergie électrique, sur site isolé à partir de ressources renouvelables, n'est pas chose aisée. En effet, en raison de la diversité des composants, et surtout du caractère très imprévisible de la ressource renouvelable, l'opération peut s'avérer délicate.

Pour effectuer le dimensionnement avec le logiciel HOMER, on peut simplement entrer une série de données et le logiciel donne la solution optimale parmi les données soumises. Ce n'est cependant pas nécessairement la solution optimale absolue, mais plutôt la solution optimale parmi les choix soumis. De plus, on n'a aucune base de comparaison pour évaluer la justesse des résultats si on ne fait aucun calcul préalable. Donc il est impérativement logique d'établir une méthodologie afin d'arriver à des résultats concluants en utilisant le logiciel HOMER.

Pour établir une méthodologie d'étude des SEH on utilise le logiciel HOMER, nous avons décortiqué la question présentée dans l'introduction en plusieurs questions :

1. Où va-t-on installer le système ?

2. Quelle sont les ressources locales du site d'installation ?

3. Quelle charge veut-on alimenter ?

4. Quels types d'équipements veut-on utiliser ?

5. Quels sont les dimensions de ces équipements ?

6. Quels sont les différents coûts des équipements à utiliser ?

7. Quels est la durée de vie du projet ?

8. Quelles sont les paramètres économiques du système ?

9. Quelle sont les contraintes à satisfaire par le système ?

10. Quelle est la stratégie de fonctionnement du système ?

11. Quelle est la configuration optimale ?

12. Quelles sont les différents coûts de la configuration optimale ?

13. Quelles sont les performances de la configuration optimale ?

14. En comparaison avec les autres configurations quelles sont les points forts et les points faibles de la configuration optimale ?

15. Est-ce que la solution trouvée reste la meilleure même s'il y a certains changements dans les différents paramètres entrés ?

Pour répondre à ces questions la démarche générale adoptée est résumée en quelques étapes :

- Présentation du site c.-à-d l'emplacement du système hybride.
- Évaluation de la ressource énergétique disponible sur le site.
- Évaluation de la demande énergétique (le profil de charge).
- Pré-dimensionnement manuel des équipements.
- Identifier les équipements nécessaires pour le système (panneaux solaires, convertisseur, batteries…) dans le logiciel HOMER.

6. Entrer les données nécessaires dans le logiciel :

- ✓ consommation énergétique

- ✓ gisement solaire pour le lieu désiré
- ✓ données des équipements
- ✓ données reliées au combustible utilisé par la génératrice.
- ✓ paramètres économiques
- ✓ données du contrôle du système
- ✓ contraintes du système

7. Lancer le calcul dans HOMER.

8. Faire des analyses sur les résultats obtenus : les dimensions, les coûts et les performances du système optimal, effectuer la comparaison avec les autres solutions en tenant compte du côté technique, économique et environnemental et faire des analyses de sensibilité.

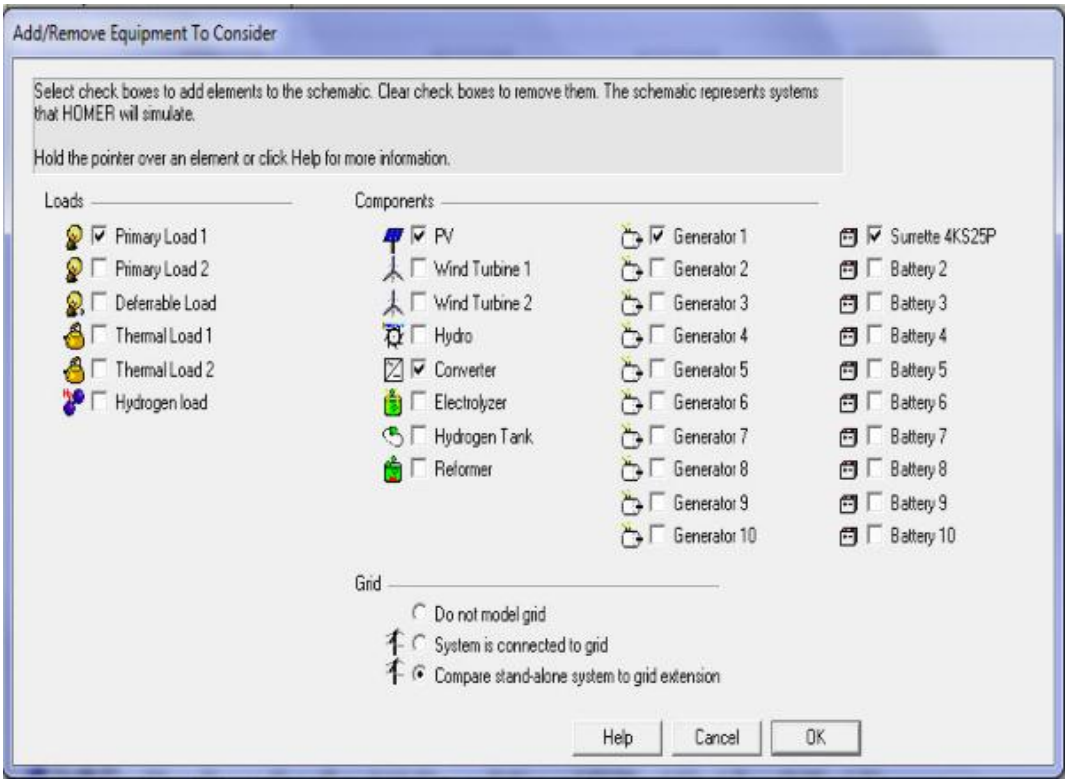


Figure.9.2. Identification des équipements nécessaires.

9.2 PVSYST :

9.2.1 Logiciel PVSYST

PVSyst est un outil de simulation centré sur le photovoltaïque (PV), développé à l'origine à l'Université de Genève. développé à l'Université de Genève mais qui

est maintenant une société indépendante. Le progiciel se concentre sur modélisation, le dimensionnement, la simulation et l'analyse des systèmes PV.

Figure. 9.3. Origine de développement de PVSyst

PVSyst dispose d'une sorte de modélisation financière, mais il s'agit principalement d'un outil de performance. Une simulation typique dans PVSyst se compose des étapes suivantes :

- Définition du projet. C'est ici que l'utilisateur crée le projet souhaité dans l'interface utilisateur, qui aussi sélectionne l'emplacement géographique et le fichier météo à utiliser. Un certain nombre de sites et de fichiers météo sont déjà inclus dans les bases de données PVSyst, mais l'utilisateur a également la possibilité d'importer ses propres fichiers. PVSyst supporte plusieurs types de fichiers météo tels que TMY2, TMY3 et EPW, ainsi que des fichiers provenant de sources telles que Meteonorm, Photoplan, etc., PVGIS (Photovoltaic Geographical Information System), le World Radiation Data Centre (WRDC), Retscreen, Helioclim et SolarGIS.

- Créer une variante du système. C'est ici que l'utilisateur crée une version de calcul du projet créé à l'étape 1. Sur l'interface, l'utilisateur peut définir différents paramètres d'entrée tels que l'orientation du module, la configuration du système et les paramètres de perte.

- Exécution de la simulation. L'utilisateur exécute la simulation et génère une variété de graphiques et de rapports pour l'analyse du système PV. l'analyse

du système PV. PVSyst permet à l'utilisateur d'analyser les résultats dans le programme, de les exporter vers un autre programme ou de sauvegarder la variante pour un usage ultérieur.

PVSyst offre à l'utilisateur des rapports et des ventilations détaillés et de précieuses informations sur les aspects techniques de la conception et du déploiement. Cela permet à PVSyst de répondre aux besoins d'un large éventail d'utilisateurs, y compris les chercheurs et les architectes. De plus, son interface est également multilingue et disponible dans d'autres langues .

Après l'avoir installé, les utilisateurs peuvent utiliser PVSyst gratuitement en mode d'évaluation pendant 30 jours durant lesquels il fonctionne à pleine capacité. Pour profiter de toutes les capacités par la suite, l'utilisateur devra acheter une licence, sinon PVSyst passera automatiquement en mode démo dans lequel il fonctionne avec des capacités limitées.

PVSyst soutient également les universitaires en leur accordant des remises sur les licences achetées à des fins éducatives. Ces remises sont disponibles pour les écoles, les universités et les autres instituts d'enseignement sur demande directe auprès de PVSyst.

PVSyst est aussi doté d'une interface utilisateur graphique qui permet aux utilisateurs non pas expérimentés en modélisation informatique de construire leurs propres systèmes PV, d'effectuer des simulations de base, et de naviguer dans le logiciel avec une relative facilité.

L'interface de PVSyst permet à l'utilisateur d'effectuer la conception et le dimensionnement de systèmes et d'obtenir des résultats sous forme de graphiques ou de tableaux, de concevoir et de dimensionner un système et d'obtenir facilement des résultats sous forme graphique ou tabulaire. Pour la plupart des tâches, l'utilisateur n'a qu'à saisir certaines valeurs requises dans les champs appropriés puis d'exécuter des actions en appuyant sur un bouton.

PVSyst documente les résultats de simulation dans des rapports, résumant les paramètres de simulation, les principaux résultats et la qualité du système. Les résultats peuvent également être visualisés sous forme de tableaux ou exportés vers un logiciel tiers tel que Microsoft Excel pour une analyse ou une présentation plus approfondie.

9.2.2 Flexibilité de la modélisation de PVSyst

PVSyst est livré sous forme d'une application de bureau complète et ne permet pas l'utilisation de scripts. Cela signifie que l'utilisateur peut uniquement modéliser un système PV dont la conception adhère à certaines normes prédéfinies dans PVSyst. Il n'est donc pas possible pour l'utilisateur d'introduire ou de modifier l'un des modèles thermiques, électriques ou optiques de PVSyst .

Les utilisateurs peuvent cependant définir des dispositifs qui ne se trouvent pas dans les bases de données de composants PVSyst en utilisant les fiches techniques des des fabricants, créant ainsi de nouveaux dispositifs dans la base de données.

9.2.3 Capacité de modélisation économique et de performance de PVSyst

PVSyst ne traite que du photovoltaïque (PV) et divise ses modèles de performance en quatre catégories:

1. Connecté au réseau: il s'agit d'un type de système qui est composé d'éléments constituant le champ photovoltaïque, c'est-à-dire des modules/chaînes de modules, onduleurs et tout ce qui va jusqu'à la connexion au réseau.

2. Autonome: un système qui doit être constitué de modules, d'une batterie et d'un régulateur.

3. Un système de pompage de l'eau.

4. Réseau DC: un système composé d'une matrice PV et d'un profil de charge destiné au réseau de transport public.

PVSyst propose également une évaluation économique pour modéliser les coûts et les investissements des projets. Avec le modèle financier, l'utilisateur peut projeter les coûts de fonctionnement et déduire la rentabilité à long terme, en particulier pour les systèmes connectés au réseau .

9.2.4 Premier contact avec PVsyst

En ouvrant PVsyst (Version 6.04), vous arrivez sur la page principale :

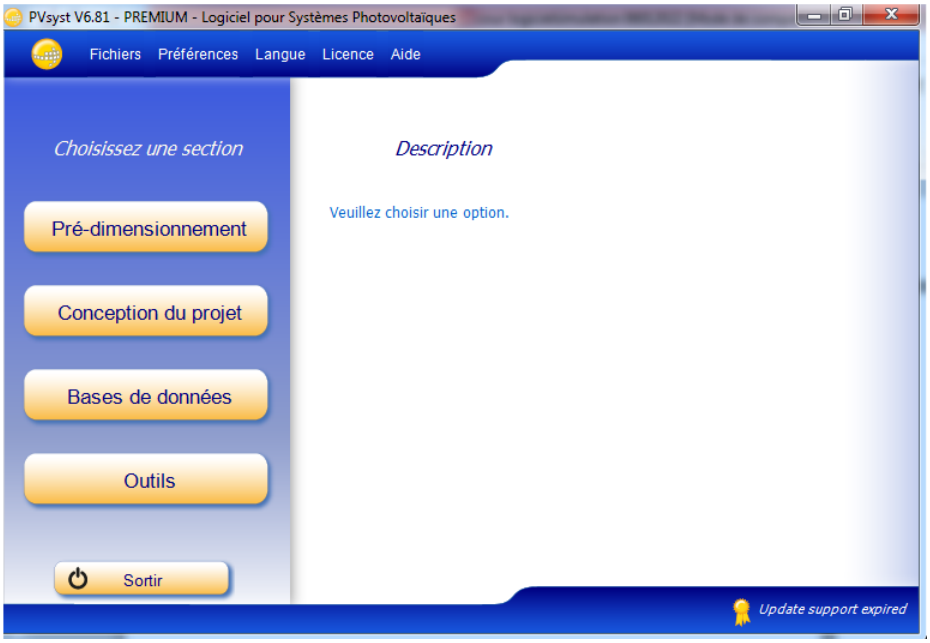


Figure. 9.4. Page principale lors du premier contact avec PVsyst .

Cela donne accès aux quatre parties principales du programme:

- ✓ **'Pré-dimensionnement'**,
- ✓ **'Conception du projet'**,
- ✓ **'Bases de données'**,
- ✓ **'Outils'**.

'Pré-dimensionnement' ne doit pas être utilisé pour un projet connecté au réseau, mais plutôt de commencer la **'Conception du projet'** complète (Fig.9.4).

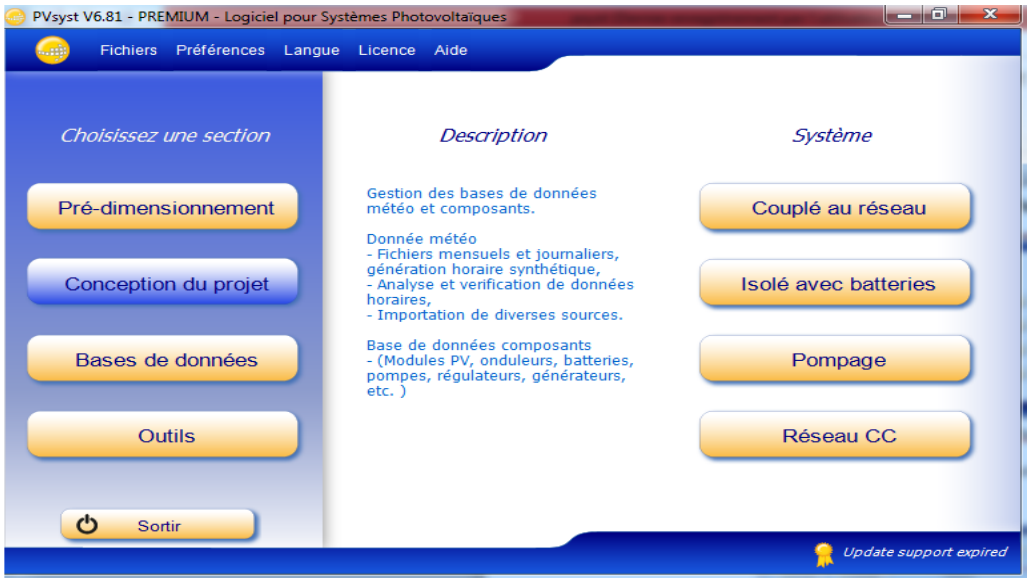


Figure. 9.5. Conception du projet .

Lorsque le projet 'couplé au réseau' est choisi, le tableau de bord suivant pour la gestion d'un projet sera obtenu (Fig. 9.10).

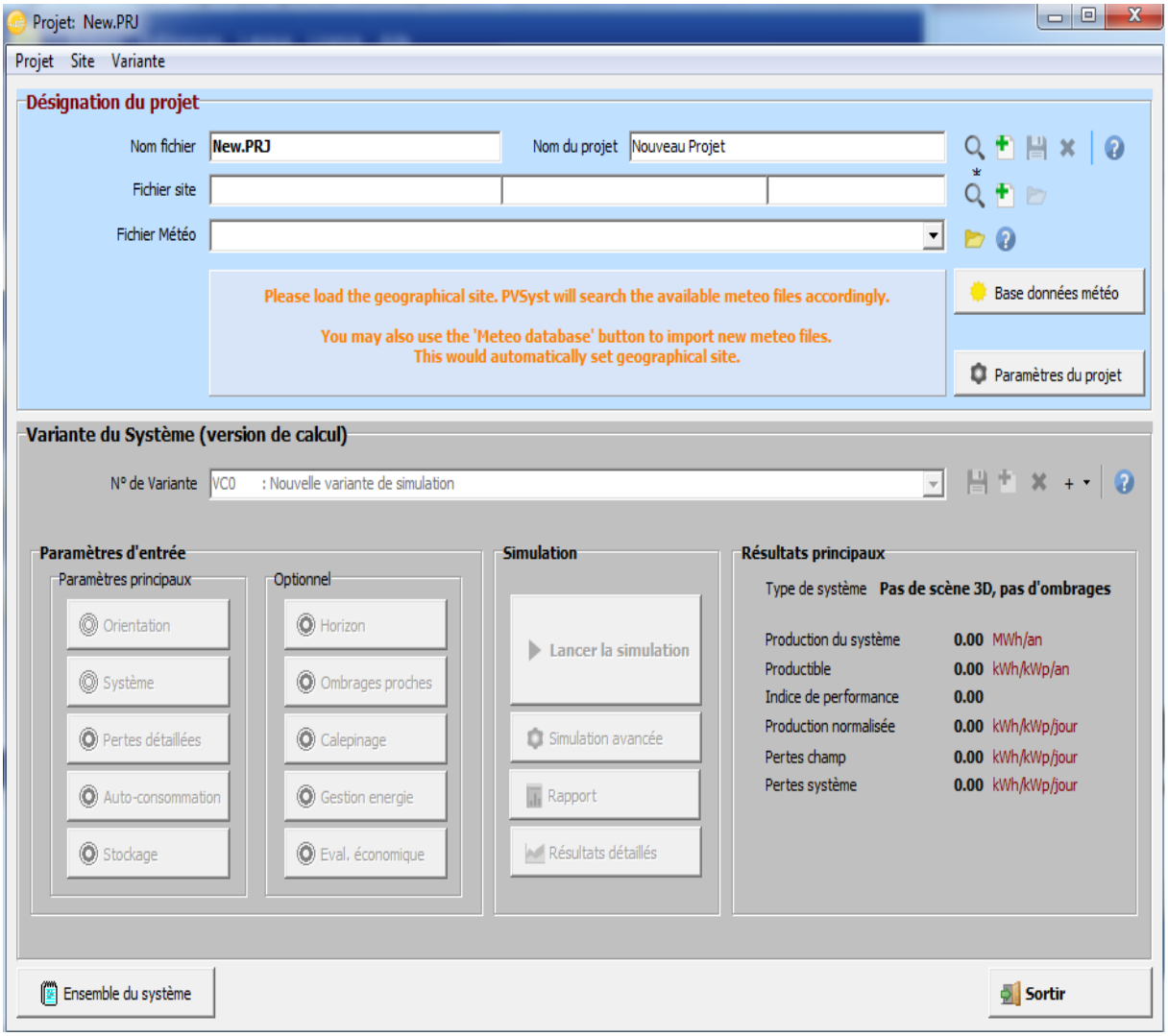


Figure. 9.10. Tableau de bord pour gérer un projet dans PVSyst .

9.2.5 Étapes du développement d'un projet

Lorsqu'on développe un projet dans PVsyst, il est conseillé de procéder par petites étapes :

- Créer un projet en spécifiant l'emplacement géographique et les données météorologiques.
- Définir une variante de base de l'installation, en indiquant l'orientation des modules PV, la puissance requise ou la surface disponible et le type de modules PV et d'onduleurs que vous souhaitez utiliser. PVsyst proposera une configuration de base pour ce choix et définira des valeurs par défaut

raisonnables pour tous les paramètres nécessaires à un premier calcul. On peut ensuite simuler cette variante et la sauvegarder. Il s'agira d'une première approximation qui sera affinée lors d'itérations successives.

- Définir des variantes successives en ajoutant progressivement des perturbations à ce premier système, par exemple des ombres lointaines, des ombres proches, des paramètres de perte spécifiques, une évaluation économique, etc. Vous devez simuler et enregistrer chaque variante afin de pouvoir les comparer et comprendre l'impact de tous les détails que vous ajoutez à la simulation.

9.2.6 Donnée système :

Contrairement au mode « preliminary design », les données systèmes seront ici concrètes. Il faut donc les informations précises sur l'onduleur et les panneaux photovoltaïques. Pour le connecté réseau, cette étape est divisée en cinq sous parties : le choix de l'onduleur "inverter", le choix du module photovoltaïque "select modules" le schéma de câblage des modules "modules array", le calcul des pertes et l'expression du besoin. Il faut donc :

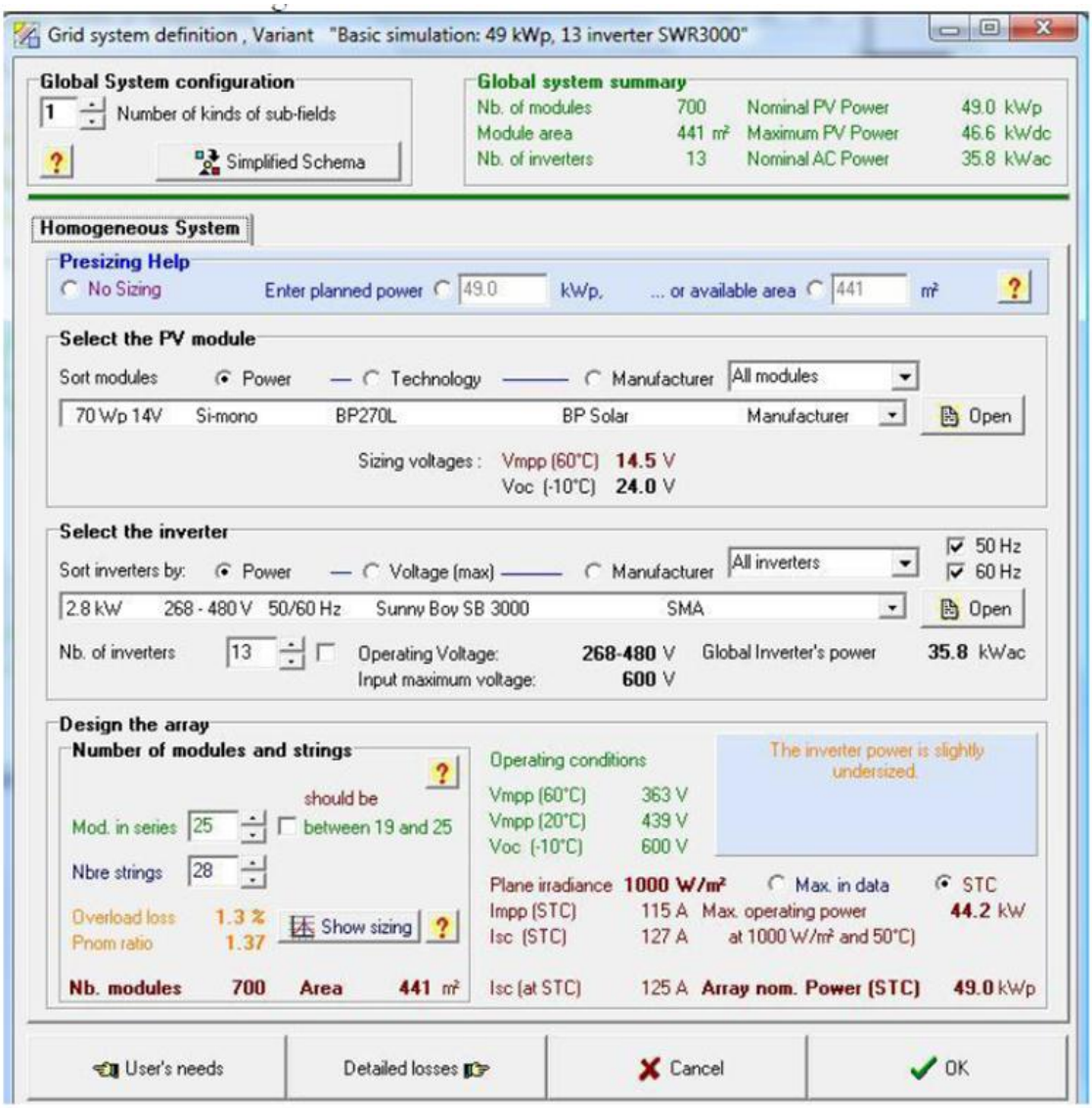
☐ Entrer le nombre de champs (si plusieurs onduleurs ou plusieurs types de panneaux) dans le cadre en haut à droite

☐ Entrer facultativement la puissance désirée nominale de l'installation ou la surface voulue dans le cadre bleu "precizing help".

☐ Choisir son onduleur : il y a possibilité de restreindre la liste en choisissant ceux encore en vente "Available Now".

☐ Choisir ses panneaux : cela fonctionne de la même manière.

☐ Rentrer le nombre de String (String = ensemble de panneaux en séries) et le nombre de panneaux par string. Si une valeur à été entré dans le cadre bleu "precizing help", le programme calculera tout seul une solution de câblage.



A noter: si l'onduleur ou panneau que vous avez n'est pas dans la liste, il faut modifier un des panneaux aux caractéristiques les plus proches. Pour cela il faut appuyer sur le bouton open à coté de la boite de sélection du produit.

Lorsque vous rentrez les caractéristiques de votre installation (nombre de panneaux/string), PVsyst vous indique si l'onduleur est « taillé » pour les strings en affichant un message qui peut être :

- ***The Array MPP operating Voltage is lower than the Inverter Minimum Operating Voltage***: L'onduleur ne reçoit pas assez de volt, il n'y a pas assez de panneaux par string.
- ***The Inverter power is slightly oversized*** : L'onduleur est surdimensionné par rapport à votre installation.
- ***The Inverter power is strongly oversized*** : L'onduleur est largement surdimensionné par rapport à votre installation.

- ***The Inverter power is slightly undersize*** : L'onduleur est sous-dimensionné par rapport à votre installation. Cela n'est pas forcement grave car la puissance maximum calculée par le logiciel n'est jamais atteinte.
- ***The Inverter power is strongly undersize*** : L'onduleur est largement sous-dimensionné, il vaut mieux en changer.
- ***The Array MPP operating Voltage is greater than the Inverter Minimum Operating Voltage****:* Il y a trop de panneaux sur votre string pour l'onduleur.

9.2.7 Simulation :

Une fois que tous les paramètres sont correctement entrés, le bouton "**simulation**" n'apparait plus grisé. Dans la fenêtre qui s'ouvre, avant de lancer la simulation, on peut choisir les données qui seront écrites dans un fichier PDF à la fin de la simulation. Pour choisir ces variables, cliquer sur "output file". On peut aussi choisir d'afficher n'importe quel type de graphe avec le bouton "special graphes".

Après la simulation, cliquer sur "results" pour accéder à toutes les données principales, et à d'autres que vous pouvez définir : graphes, graphes personnalisés, tableau personnalisés, données économique et le rapport au format PDF en appuyant sur "report".

REFERENCES

Références

[1] A. Lonton « Méthodes et outils dela simulation », Edition, Hermès, 2000

[2] Marie Postel « Introduction au logiciel MATLAB » version réservé , septembre 2004

[3] Support de cours, « Outils de Programmation pour les Mathématiques» reéalisé par : Dr MOHAMMED OMARI, Université ADRAR

[4] Nadia Martaj · Mohand Mokhtari « MATLAB R2009,SIMULINK et STATEFLOW pour Ing´enieurs, Chercheurs et Etudiants »

[5] Support de cours, « Le calcul scientifique appliqué au Génie Civil Sous MATLAB » reéalisé par : BABA HAMED FATIMA ZOHRA, USTO

[6] HAHN, Brian et VALENTINE, Daniel « Essential MATLAB for Engineer and scientists » Newnes 2007

[7] Ziad Manssouri « cour langage » , Université de Skikda 20 août 55, Département de Technologie

[8] Yassine Ariba et Jérome Cadieux « Manuel MATLAB » version 0.1, Départements GEI & Mécanique, Icam - Toulouse

[9] Choubane Faiza « Initiation au language de programation Matab »Edition 2016 , ISBN : 978-9947-35-058-4.

[10] AFANOU Sitou Dela-Dem, CHAIBOU Halidou, BOUARE Karim « Simulation des Machines A Courant Continu dans l'environnement Matlab/Simulink »Cycle Ingénieur - 3ème Année Electricité ©Groupe A3 - Mai 2011

[11] M.Djebli, H.Djelouah « MAtlab cours et exercices corrigée » pages bleus ISBN :978-9947-34-103-2

[12] M. Djebli, S.Idjimarene« Méthodes numériques Application avec MATLAB» ISBN :978-9974-34-091-1

[13] Support de cours, « Langage du calcul scientifique (Matlab) » université Abderrahmane Mira de Bejaia

[14] Offre de formation L.M.D « Licence Académique » Programme national 2018-2019

[15] Manfred Gilli « Méthode numériques » Université de Genève Version 25 mars 2006

[16] B. AFIF, A. Chaker, A. Benhamou "Sizing of Optimal of Standalone Hybrid Power System Using Homer Software ",International Review of Automatic Control (I.RE.A.CO.), Vol. 10, N. 1, / ISSN 1974-6059, Jannary2017

[17] B. AFIF, T. Allaoui, A. Chaker, A. Benhamou"Numerical Simulation of Solar on Grid Plan in Tamanrasset",International Review of Automatic Control (I.RE.A.CO.), Vol. 8, N.3 / ISSN 1974-6059, May2015

SITE WEB

[1] www.homerenegy.com

[2] www.Pvsyst.com

[3] http://www.mathworks.com/